Faculty of Humanities
Curtin University Sustainability Policy Institute

Ultra-efficient Bus Rapid Transit Timetabling

Matthew John Bradley

This thesis is presented for the Degree of
Doctor of Philosophy
of
Curtin University of Technology

June 2010

< blank page >

< blank page >

< blank page >

< blank page >

< blank page >

Declaration and Copyright Notice

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgement has been made.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature: .....

Date: .....

< blank page >

< blank page >

< blank page >

< blank page >

< blank page >

Abstract

Bus Rapid Transit (BRT) systems are increasingly used, particularly in the developing world, to provide low-cost, high-capacity urban mobility. An example of this trend is Bogotá's TransMilenio BRT system, the test site for this thesis, which uses an homogeneous fleet of 18 metre long articulated buses to service 1,450,000 passenger trips per day, and which reaches a peak passenger load level of 45,000 passengers per hour per direction. The computational tools and techniques used to plan the timetables of such BRT systems are largely the same set of tools and techniques used to plan non-BRT transit systems.

Unlike other transit systems, high-load BRT systems commonly run simultaneous express services, a situation that the tools developed to timetable non-BRT transit systems were not specifically designed for. Due to the running of simultaneous express services, the timetabling of high-load BRT systems becomes a combinatorial problem, a far more complex class of problem than normal non-combinatorial timetabling. The thesis is advanced here that high-load BRT systems could be timetabled far more efficiently via a software tool built around a recognition of the problem's combinatorial nature.

This thesis is tested by building such a software tool, using that tool to develop an alternative timetable for the Américas Line of the TransMilenio BRT system, and then comparing that timetable's performance to the performance of the existing timetable. Data used for the Américas Line was from the period Monday the 23rd to Friday the 27th of May 2005. The software tool developed to process this data changes

express service stopping patterns as quickly as passenger load changes, leading to a great many express patterns over the course of a one day timetable. Due to this rapid tracking of passenger load, the resulting timetable is referred to as an “ultra-efficient timetable.”

The ultra-efficient timetable produced for the Américas Line has 88 unique stopping patterns, compared to the existing timetable’s three, and is shown to be tracking passenger load far more precisely. Bus fleet size under the ultra-efficient timetable is 9% lower than for the existing timetable, indicating an estimated capital cost saving for the Américas Line of US$1.8 million. Bus kilometres travelled under the ultra-efficient timetable are 40% lower than for the existing timetable, indicating an estimated annual operating cost saving for the Américas Line of US$6 million.

The ultra-efficient timetable delivering these performance improvements is shown to be approximately ten times more complex than could be reasonably deployed using paper timetables. Consequently, an ultra-efficient timetable would need to be deployed in conjunction with a fully-automated passenger information system. With this caveat, the thesis that high-load BRT systems can be far more efficiently timetabled using a combinatorial software tool is confirmed here. As an alternative to deploying ultra-efficient timetables, the combinatorial timetabling technology developed for this thesis could also be used to produce more efficient versions of normal paper-based BRT timetables.

Acknowledgements

Professor Jeff Kenworthy
Supervisor 2004-2006, 2008-2010

Professor Tom Lyons
Supervisor 2007

Dr Jan Scheurer
Supervisor 2007

TransMilenio S.A.
Research Site and Data Provider

Dr Rolf Bergmaier: For sharing with me his knowledge of clock-face timetabling., Eric Britton: For assisting me to acquire BRT data., Gino Brunetti: For organising the translation of documents for me., Alexandre Greff Buaes: For translating documents for me., Francois Carignan: For the time he spent corresponding about a meeting., Angelica Castro: For organising for me to be provided with TransMilenio BRT data., Professor Avishai Ceder: For sharing with me his knowledge of timetabling and scheduling., Oscar Edmundo Diaz: For assisting me to acquire BRT data., Cleon Ricardo dos Santos: For assisting me to acquire BRT data for Curitiba, Brazil; data that, as it turned out, did not end up being used., Marcos Ricardo dos Santos: For assisting me to acquire BRT data., Marc Dupont: For the time he spent corresponding about a meeting., Kimberley Emmerson: For the time she spent corresponding about a meeting., Charles Fleurent: For sharing with me his knowledge of timetabling and scheduling., Paul Hamelin: For taking the time to meet with me to discuss timetabling.,

Professor David Hensher: For sharing with me his knowledge of timetabling., Enrique Hernandez: For organising the collation of TransMilenio BRT data for me., Dario Hidalgo: For sharing with me his insights into the early development of the TransMilenio., Felix Laube: For assisting me to acquire BRT operation data for Bern, Switzerland; data that, as it turned out, did not end up being used.,
Professor Luis Antonio Lindau: For assisting me to contact the Porto Alegre BRT system operators., Alejandro Niño: For sharing with me his knowledge of the TransMilenio BRT., Carlos F. Pardo: For being a welcoming and informative host in Bogotá., Michael W. Roschlau: For assisting me in contacting experts in the timetabling field., Sebastian Pena Serna: For translating documents for me., Eduardo A. Vasconcellos: For assisting me to acquire BRT data.

The Public Transport Authority of Western Australia: For providing me with passenger data for the Perth railway system.

Inclusion of a person's or organisation's name in this acknowledgements section should not be taken as an indication of their support for this thesis, its content, or conclusions.

Table of Contents

< blank page >

< blank page >

< blank page >

< blank page >

< blank page >

List of Figures

List of Tables

< blank page >

< blank page >

< blank page >

< blank page >

< blank page >

List of Plates

< blank page >

< blank page >

< blank page >

< blank page >

< blank page >

Chapter 1

Introduction

1.1 Background to BRT Systems and Thesis Rationale

Bus Rapid Transit (BRT) is a relatively new transit technology, with the first BRT system, the Curitiba BRT, being opened in 1974.[1] The central concept of BRT is to use buses to provide a transit service comparable to that of a passenger rail service, but at a far lower capital cost. To provide a comparable service, the running ways used by BRT systems are, as with passenger rail systems, either partially or fully segregated from general traffic. In its global survey of BRT systems, the 2007 *Bus Rapid Transit Planning Guide* does not list a single BRT system not running on at least a partially segregated roadway.[2] Other features common to passenger railway systems, such as at-level boarding, relatively wide stop spacing and pre-boarding ticketing, are also common features of BRT systems.[3] One paper by the Canadian Urban Transit Association simply defines BRT as “a rubber tired rapid transit system with running ways, stations, and all the other attributes normally associated with rail rapid transit.”[4]

At this time BRT systems are not very widely used, with the global survey in the 2007 *Bus Rapid Transit Planning Guide* listing only 44

1 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York. pp. 22

2 op. cit., pp. 767-778

3 op. cit., pp. 754-778

4 McCormick Rankin Corporation (2004), *Bus Rapid Transit: A Canadian Industry Perspective*, Canadian Urban Transit Association, Toronto , p. 15

systems.[5] BRT is, though, a growth area, both in the traditional region of the world for BRT systems, Latin America, and in non-traditional areas, particularly in Asia. An example of the expansion of BRT systems in Latin America is the opening, in 2009, of a second line on Mexico City's Metrobús BRT system.[6] In Asia, the construction of a high-load BRT line was begun in the Chinese city of Guangzhou in 2008; a line that is expected to be opened by the end of 2009.[7] Furthermore, according to an article by the Institute for Transportation and Development Policy "in more than a dozen Chinese cities, efforts are underway to develop Bus Rapid Transit Systems as a competitive travel option."[8] The applicability of the research presented in this thesis will grow along with the number of BRT systems.

As suggested by the above brief background, BRT systems are increasingly used to provide low-cost mass transit, particularly in the developing world. For urban transport managers, BRT provides the twin benefits of being relatively inexpensive and relatively quick to deploy. For example, the first 41 kilometre route of the TransMilenio BRT system in Bogotá, Columbia — the test site for this thesis — was built between

5 op. cit., pp. 767-778

6 *Mexico City Opens Second Metrobús Line* (2009), EMBARQ: The World Resources Institute Centre for Sustainable Transport Website, http://www.embarq.org/en/news/09/01/28/mexico-city-opens-second-metrobús-line (Accessed 10 February 2009)

7 *Guangzhou BRT Construction Begins: by Karl Fjellstrom* (2008), The Institute for Transportation & Development Policy Website, Available: http://www.itdp.org/index.php/projects/update/guangzhou_brt_construction_begins/ (Accessed 12 July 2009)

8 *BRT Poised for Take-Off in China: by The Institute for Transportation & Development Policy* (2005), The Institute for Transportation & Development Policy Website, Available: http://www.itdp.org/STe/ste19/brt.html (Accessed 24 January 2006)

1998 and early 2002[9] at a tenth the cost of the heavy rail alternative.[10] Early BRT installations have mostly been in Latin America, and 7 of the 10 current largest systems are Latin American systems, measured by passenger trips per day.[11]

Bogotá's TransMilenio BRT system is the second highest load system, measured by passenger trips per day, but ranks as the highest load system in terms of passengers per hour per direction (pphpd), with a peak load of 45,000 pphpd.[12] The next three highest load systems are the São Paulo Interligado with a peak load of 34,900 pphpd, the Porto Alegre BRT with a peak load of 28,000 pphpd and the Curitiba BRT with a peak load of 20,000 pphpd, all of which are Brazilian systems.[13] The last of these systems, the Curitiba BRT, which opened in 1974,[14] has been the model for many of the other BRT systems, demonstrating, in practice, that high loads can be serviced by a bus-based transit mode. Enrique Peñalosa, the mayor who, in 1997, proposed a BRT system for Bogotá, wanted "a bus-based mass transit system modelled on those in operation in the Brazilian cities of São Paulo and Curitiba."[15] A mere ten years later and the TransMilenio BRT is the highest load BRT system in the world, as measured by passengers per hour per direction.[16] This outcome has not

9 Cain, A. (Principal Investigator) 2006, *Applicability of Bogotá's TransMilenio BRT System to the United States - Final Report May 2006*, Federal Transport Administration, Department of Transportation, United States of America, Washington, D.C., U.S.A, p. vi

10 op. cit., p. xii

11 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York. pp. 767-778

12 op. cit., p. 767

13 op. cit., pp. 767-778

14 op. cit., p. 22

15 *Q&A With Darío Hidalgo:* by Erico Guizzo (2009), IEEE Spectrum Website, Available: http://www.itdp.org/STe/ste19/brt.html (Accessed 17 January 2009)

16 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York. pp. 767

gone unnoticed, with one article observing that “when it comes to moving lots of people on buses, transportation experts are quick to mention one particular project these days: the TransMilenio.”[17]

The TransMilenio, and other high-load BRT systems, commonly service high passenger loads by running express bus services at the same time as an all-stop bus service. It is not unusual for more than one express service to be run at the same time as an all-stop service, with, for example, the Américas Line of the TransMilenio BRT system running up to three express services at the same time as an all-stop service.[18] This regular running of express services, side-by-side with an all-stop service, is an unusual feature of high-load BRT systems, resulting from the need to service passenger loads of tens-of-thousands of passengers per direction per hour, with buses rated to carry, in the TransMilenio’s case, only 160 passengers.

Choosing how many express services to run, and selecting which stops those express services are to stop at, is part of the transit planning process, where transit systems are modelled so that timetables and other operational data can be determined. With regards to BRT systems, Dr. Dario Hidalgo, Deputy General Manager of the TransMilenio BRT from June 2000 to September 2003,[19] views the modelling and simulation process as being . . . very important.

17 *Q&A With Darío Hidalgo: by Erico Guizzo* (2009), IEEE Spectrum Website, Available: http://www.itdp.org/STe/ste19/brt.html (Accessed 17 January 2009)

18 From bus timetable data received by the author from the TransMilenio S.A.

19 *Q&A With Darío Hidalgo: by Erico Guizzo* (2009), IEEE Spectrum Website, Available: http://www.itdp.org/STe/ste19/brt.html (Accessed 17 January 2009)

He states:

> “Knowing your numbers before implementation can be key to the success of the system. In terms of modelling the transport of passengers, this is an area where research has reached a mature level. The tools are very advanced, although there could be better ones for public transportation projects. In complex systems like the TransMilenio, simulation may be the only way to go in terms of learning how to solve the bottlenecks and improve operations. You don’t want to try certain changes in real life because you could create serious trouble.”[20]

Although some of the modelling tools used by BRT planners are indeed very advanced, this is not the case in the area of express service stopping pattern selection. As will be discussed in this thesis, there is no evidence of even a semi-automated software tool to assist with express service stopping pattern selection for BRT systems. The lack of such a software tool is not particularly surprising given that BRT is still a fairly new transit mode, and that the use of express services, running at the same time as an all-stop service, is an approach not widely seen outside of a limited number of high-load BRT systems.

1.2 Thesis Aim and Questions

This thesis will investigate the possibility that more efficient BRT timetables might be produced using a purpose designed software tool. To

20 ibid.

assess the extent of any efficiency gap that might exist between a BRT timetable produced using current express service timetabling practices, and one produced by a purpose designed software tool, such a software tool needs to be built and then tested against current practices. This is done here using one line of Bogotá's TransMilenio system as the test site and source of data. The objective of this thesis is to consider just how efficient a non-live[21] BRT timetable might be able to be, if it were constructed using a purpose designed software tool. The existing TransMilenio timetable will be used to assess the magnitude of any difference between the to be developed ultra-efficient timetable and a 'normal' timetable. The TransMilenio's 'normal' timetable will be referred to as an "efficient timetable."

In order to conduct the broadly outlined research above, a series of carefully considered steps have to be undertaken. These steps involve answering a series of questions that build logically upon one another. The steps in conducting this research are now outlined along with the key research questions that are to be addressed in this thesis.

Firstly, before embarking on this research it is necessary to understand clearly what the literature reveals about the nature and characteristics of BRT systems. So it is necessary as the initial step to address the question:

21 A non-live timetable is one developed for a transit system before being deployed, using expected, rather than actual, passenger load data. The alternative to this approach would be to timetable a transit system live, by assessing the current passenger loads on the transit line, and organising the current vehicle resources on the line, minute-by-minute, to best service those loads.

> Is there a clear definition of what is, and what is not, a BRT system in the literature?

If it is not possible to clearly say what a BRT system is, one cannot, after all, meaningfully talk about “Ultra-efficient BRT Timetabling.”

Once this question has been addressed, a key question about timetabling needs to be examined, namely:

> Is there any theoretical evidence that current manual timetabling techniques are ill-suited to the task of producing highly efficient BRT timetables?

Simply because there is not an automated method for producing BRT timetables does not mean that the current manual methods are necessarily inefficient; manual does not necessarily imply inefficient any more than automated necessarily implies efficient.

Following up a theoretical assessment of the efficiency of current BRT timetabling techniques, with an assessment of the efficiency of those techniques in practice, requires analysing data from a real-world BRT system. The real-world BRT system test site for this thesis is the Américas Line of the TransMilenio BRT system. To assess how efficiently this one line of the TransMilenio is being timetabled using current manual methods, the data for the Américas Line will need to be separated from the TransMilenio system as a whole. Thus the next question that has to be answered is:

> Can the data for one line of a BRT system be easily separated from the data for the entire system?

Not only is an isolated BRT line necessary for the production of an ultra-efficient timetable, it also makes it possible to address the question:

> Are current BRT systems timetabled in a highly efficient fashion?

This question is the real-world parallel to the earlier question asking whether there is any theoretical evidence that current manual timetabling techniques are ill-suited to the task of producing highly efficient BRT timetables. Determining whether current BRT systems are timetabled in a highly efficient fashion will allow an assessment to be made as to exactly how much ‘efficiency room’ is left above current efficient BRT timetables for a separate class of timetables, ultra-efficient timetables, to occupy.

Precisely locating ultra-efficient timetabling, on a spectrum of timetabling efficiency, requires knowing not only that it is to be located ‘above’ current efficient timetabling, in terms of efficiency, but also to investigate what it might be ‘below’ in terms of efficiency. If there is no class of timetabling above ultra-efficient timetabling, then ultra-efficient timetabling is, in fact, optimally-efficient timetabling. This issue will be investigated by asking:

> Is it possible to generate optimally efficient timetables for BRT systems?

Just how ultra-efficient timetables might fit into whatever ‘efficiency room’ exists between current timetabling practices and a notional optimally-efficient timetabling practice, requires the concept of ultra-efficient timetabling to be clearly defined. It is therefore necessary to ask:

> Is it possible to develop a definition of ultra-efficient BRT timetabling, one that clearly differentiates it from current BRT timetabling?

In the same manner that one cannot meaningfully talk about “Ultra-efficient BRT Timetabling” without being able to clearly say what BRT is, to meaningfully talk about the topic of this thesis it is also necessary to be able to clearly say what ultra-efficient timetabling is. In particular, without a clear definition of ultra-efficient timetabling, it is difficult to see how a method could be developed for the production of ultra-efficient timetables.

A key component of any ultra-efficient timetabling method, will be a metric with which to judge the performance of different express service stopping patterns. It is therefore important to ask:

> Can a simple and effective metric to judge the efficiency of BRT express service stopping patterns be easily devised?

It is proposed that a metric to judge the efficiency of BRT express service stopping patterns, embedded at the centre of a purpose designed software tool, will allow for the production of an ultra-efficient timetable for the Américas Line.

Following this logical line of inquiry and questions, which shape the research to this point, and by comparing this ultra-efficient timetable to the existing efficient timetable, the central question to be answered by the research in this thesis becomes:

> *Can a BRT timetable which varies its service frequencies and stopping patterns at the same pace that passenger load levels and patterns change, significantly outperform a normal BRT timetable?*

To rephrase the above question as a thesis, *the thesis to be tested is: whether a BRT timetable which varies its service frequencies and stopping patterns at the same pace that passenger load levels and patterns change, can significantly outperform a normal BRT timetable.* This thesis will, in due course, either be confirmed or rejected. All of the research questions that have led up to this central question about the performance of ultra-efficient timetables have been directed at enabling this question to be answered, and all the research questions to follow after, are consequent to it.

The most important question consequent to the previous research question about the performance of ultra-efficient timetables is:

> Given the likely greater complexity of any ultra-efficient timetable, would it be possible to realistically deploy such a timetable without a fully-automated passenger information system?

Based on a knowledge of the complexity of existing timetables that mix all-stop services with a variety of express services, it is an untested assumption of this research that it would not be possible to realistically deploy an ultra-efficient timetable without a fully-automated passenger information system. By conducting the entire ultra-efficient timetable development process, the validity of this assumption will be able to be tested.

Logically and finally, conducting the entire ultra-efficient timetable development process, will also make it possible to ask:

> Could any of the techniques developed to produce ultra-efficient BRT timetables be applied to improve the efficiency of current ‘normal’ paper-based BRT timetables, while keeping those timetables as paper-based timetables?

1.3 Scope and Approach of the Research

Providing useful answers to the research questions detailed in the previous section will require an interdisciplinary approach. First and foremost this is a transportation studies thesis, as evidenced by the fact that the two most heavily cited works, *Urban Transit Systems and*

Technology[22] and the *Bus Rapid Transit Planning Guide*,[23] are both transportation studies' works. Transportation studies is itself, though, an interdisciplinary area of research. It is an issue, to use the old fashioned vernacular, of men and machines: in modern times transportation is largely about machines moving men, and so to understand transportation it is necessary to understand both men and machines. Understanding how the machines move requires input from disciplines such as physics and engineering, whereas understanding the people who are moved by the machines requires input from disciplines such as psychology and sociology. To understand the people and machines together, a functioning transportation system, requires further input from disciplines such as accounting, economics, information technology and operations research.[24]

For an interdisciplinary research project, the scope of research is bounded by the domain of the problem being studied, rather than by the domain of any given discipline. This thesis reports on research undertaken to examine the problem of timetabling BRT systems in an ultra-efficient manner, and so the contours of this problem are the boundaries of the research presented here. Along with a core of transportation studies concepts, this thesis draws significantly on concepts and techniques from the computer science discipline, which will be used to build a purpose designed BRT timetabling software tool. Furthermore, as one of the core issues of ultra-efficient timetabling is

22 Vuchic, V. R. (2007), *Urban Transit Systems and Technology*, John Wiley & Sons, New Jersey

23 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York

24 Operations research is the study of complex systems, normally for the purpose of making those systems operate more efficiently. Operations research is usually considered to be a branch of applied mathematics.

efficiency, some concepts from the accounting and economics disciplines, disciplines that have much to say about efficiency, are to be found in this thesis. Finally, other disciplines, such as operations research, philosophy and sociology, are also drawn on, in smaller measure, as needed.

The specific disciplinary mix, just how much of each particular discipline's concepts and techniques are to be used, depends on the approach taken to the research. As indicated by the research questions, the approach to be taken in this thesis will be to examine the concept of ultra-efficient BRT timetabling using real test site data, and using a real current BRT timetable as a gauge against which to measure performance. This approach has the key benefit of allowing an assessment to be made of the performance of an ultra-efficient BRT timetable in a real-world situation. There are, though, significant costs to using this real-world approach. Specifically, a real-world approach requires the use of real-world data and the development of an ultra-efficient timetabling software tool. Using real-world data is a significant cost because acquiring data from operators can be a very time consuming process, and also because, once acquired, the operational data often needs significant conversion to be usable for research purposes. Writing a software tool is a significant cost because software production is a difficult and time consuming task.

The topic of this thesis could be researched in purely theoretical terms, without building a software tool to generate a full 'alternate' timetable. Furthermore, such purely theoretical research could be conducted using constructed rather than real-world data. Choosing to take a real-world, rather than theoretical, research approach significantly increases the difficultly of the research to be undertaken here. Further

increasing the difficultly level was consequently considered unwise, and therefore the simplest feasible approaches are to be applied in other areas. Specifically, in two areas, computational optimisation and behavioural analysis, the research in this thesis will use simple rather than sophisticated techniques.

With regards to computational optimisation, if one's objective is to 'squeeze every last drop' of efficiency out of a given complex system, then the techniques of the operations research discipline are the ones to apply. As the research questions made clear, though, ultra-efficient timetabling is not necessarily the same thing as optimally-efficient timetabling; there may well be a great deal of room between current practices and a notional optimally efficient practice. Furthermore, the techniques of the operations research discipline are far from simple ones.

For example, the Ph.D. thesis of Dr. Dennis Huisman, *Integrated and Dynamic Vehicle and Crew Scheduling*,[25] is a pure operations research thesis, the overwhelming bulk of which is spent discussing operations research techniques as they apply to the topic of the thesis. Although complex mathematical formulations run throughout Dr. Huisman's thesis, only very minor software development is evident in the thesis.[26] Furthermore, in the area of vehicle and crew scheduling, it would appear that real-world data, in a form appropriate for researching this topic, was readily available; Dr. Huisman's thesis spends only one page discussing the "properties of the real-world data instances" he is using, and there is

25 Huisman, D. (2004), *Integrated and Dynamic Vehicle and Crew Scheduling*, Ph.D. Thesis, Erasmus University Rotterdam, The Netherlands

26 op. sit., pp. 49-50

no discussion of the real-world data needing to be transformed in any manner.[27]

The problem of ultra-efficient BRT timetabling could have been tackled as a pure operations research problem. Such an approach, though, would likely have either required that the problem be tackled in far more theoretical terms, or for the scope of the research to be expanded beyond the normal limit for a Ph.D. thesis. As Dr. Huisman's thesis demonstrates, with regards to highly complex problems, operations research techniques are too complex to be applied in a minor way. The sophisticated techniques of the operations research discipline are, consequently, not directly utilised in this thesis.

Similarly, on the human side of researching ultra-efficient BRT timetabling, the sophisticated techniques of behavioural analysis have also not been utilised in this thesis. A key issue with ultra-efficient timetabling is people's ability to cope with timetables of varying complexity: how are people likely to behave when faced with a timetable of a given complexity, and how might that behaviour be managed or modified? With regards to transportation studies, Dr. Susan Shaheen used behavioural analysis, and techniques from related domains of knowledge, to study behavioural adaptation to a car-sharing system. In Dr. Shaheen's Ph.D. thesis, *Dynamics in Behavioural Adaptation to a Transportation Innovation: A Case Study of Carlink – A Smart Carsharing System*,[28] "social learning and social marketing theories were used . . . to

27 op. sit., pp. 90-91

28 Susan S. (2004), *Dynamics in Behavioural Adaptation to a Transportation Innovation: A Case Study of Carlink – A Smart Carsharing System*, Institute of Transportation Studies, Davis, California

explain the processes by which travelers can and might accept or adapt to a transportation innovation."[29]

Obviously such techniques might well have great utility to study the potential 'transportation innovation' of ultra-efficient timetabling, particularly with regards to how people interact with timetables as the complexity of those timetables increases. As with operations research though, proper application of behavioural analysis techniques in the context of a complex problem is a task for a whole thesis, not something to be applied in a minor way; the behavioural and attitudinal questionnaires alone in Dr. Shaheen's thesis take up over 60 pages.[30] Consequently, the sophisticated techniques of behavioural analysis will not be utilised in this thesis.

The decision to use real-world data and to build a real prototype software tool with which to trial ultra-efficient timetabling, significantly expanded the scope of the research for this thesis. To balance this, to keep the scope of the research within reasonable bounds, the simplest effective optimisation techniques are to be used and the minimum necessary assessment of customer behaviour is to be made. Insights are drawn from the operations research discipline and its techniques, and the use of operations research techniques is the first item listed in the suggestions for further research section, but operations research techniques are not directly applied. Similarly, consideration is given to the behavioural question of just how complex a timetable can be while still being understood, but behavioural analysis techniques are not directly applied. This thesis is not an operations research or behavioural

29 op. sit., p. 6

30 op. sit., pp. 340-408

analysis thesis, it is an interdisciplinary transportation studies thesis, with a large computational modelling component.

1.4 Thesis Structure

Chapter two of this thesis will introduce BRT systems and the process of timetabling. The objective of this chapter will be to provide all of the background information necessary to understand the research to be presented and to review the current state of knowledge. Bus Rapid Transit will be introduced by examining the defining characteristics of BRT systems in general, and then by looking at some specific characteristics of the TransMilenio BRT system, the test site for this thesis. Current timetabling methods will then be introduced as part of the overall task of planning a transit system's operation. It will be highlighted that one part of this planning process, the scheduling of vehicles, is solved, most efficiently, via the use of purpose designed software tools. Similarities between vehicle scheduling and the special case of high-load BRT timetabling will be explored, and the proposition advanced that high-load BRT timetabling might also be most efficiently solved via the use of a purpose designed software tool.

Chapter two thus addresses the first question posed by this thesis, that is whether there is a clear definition of what is, and what is not, a BRT system in the literature. In addition, chapter two addresses the second question posed by this thesis, that is whether there is any theoretical evidence that current manual timetabling techniques are ill-suited to the task of producing highly efficient BRT timetables.

Chapter three of this thesis will examine the Américas Line of the TransMilenio BRT system and the techniques available to isolate the Américas Line data from the data of the TransMilenio system as a whole. To provide a context for this data isolation task, the physical characteristics of the Américas Line, such as line length and stop spacing, will be presented.

Chapter three addresses the third question posed by this thesis, that is whether the data for one line of a BRT system can be easily separated from the data for the entire system. In addition, chapter three addresses the fourth question posed by this thesis, that is whether current BRT systems are timetabled in a highly efficient fashion.

Chapter four of this thesis will discuss the concept of ultra-efficient timetabling and will develop a method with which BRT systems can be timetabled in an ultra-efficient manner. A number of characteristics of ultra-efficient timetabling will be discussed, and, as part of this discussion, the question of whether BRT systems can be timetabled in an optimally efficient manner will be addressed. The identified characteristics of ultra-efficient timetabling will be used as a basis from which to develop a definition of ultra-efficient timetabling, one that clearly distinguishes it from normal efficient timetabling. From this definition of ultra-efficient timetabling a method, a set of steps, will be developed that can be followed to timetable a BRT system in an ultra-efficient manner.

Chapter four therefore addresses the fifth question posed by this thesis, that is whether it is possible to develop a definition of ultra-efficient BRT timetabling, one that clearly differentiates it from current

BRT timetabling. In addition, chapter four addresses the sixth question posed by this thesis, that is whether it is possible to generate optimally efficient timetables for BRT systems.

Chapter five of this thesis describes the operation of the software tool that will be used to generate an ultra-efficient BRT timetable for the TransMilenio's Américas Line. Specifically how each part of the software tool works, down to the level of how it actually organises and transforms data, will be covered in plain English. At the centre of the to be developed software tool will be a metric used to judge the efficiency of BRT express service stopping patterns.

Chapter five thus addresses the seventh question posed by this thesis, that is whether a simple and effective metric to judge the efficiency of BRT express service stopping patterns can be easily devised.

Chapter six of this thesis will present and analyse the ultra-efficient timetable that results from using a purpose designed software tool to process the Américas Line data. The resulting ultra-efficient timetable will be assessed using various metrics.

Chapter six addresses the final three questions posed by this thesis, questions eight, nine and ten. The eighth question posed by this thesis is whether a BRT timetable which varies its service frequencies and stopping patterns at the same pace that passenger load levels and patterns change, can significantly outperform a normal BRT timetable. The ninth question posed by this thesis is whether, given the likely greater complexity of any ultra-efficient timetable, it would be possible to realistically deploy such a timetable without a fully-automated passenger information system? The tenth and final question posed by this thesis is

whether any of the techniques developed to produce ultra-efficient BRT timetables could be applied to improve the efficiency of current ‘normal’ paper-based BRT timetables, while keeping those timetables as paper-based timetables.

In summary, chapter one has established the background and rationale of the thesis and the key research questions to be answered. It has also explained the boundaries of the research, the disciplines involved, its intended scope and the approach taken to answer the questions. It has clearly delineated what the thesis will attempt to do and, importantly, what it will not attempt to do. Finally, chapter one has explained the structure of the thesis. In summary terms, chapter two is a literature review that identifies gaps in knowledge surrounding the topic of BRT timetabling and provides the foundation for the original research to be conducted to help fill those gaps. Chapters three, four and five provide a systematic and detailed methodology of how the research in this thesis will be carried out. As the thesis is mostly concerned with the development of a custom software timetabling tool, these three chapters are the core of the thesis and will answer many of the research questions posed in this thesis. Chapter six will present the results of the research, and, in doing so, answer the central thesis research question concerning the efficacy of ultra-efficient timetabling. Finally, chapter seven provides a summary of the findings of the thesis, in terms of answers to each of the research questions addressed in each chapter. Chapter seven will conclude by presenting a series of suggestions for further research.

Chapter 2

Bus Rapid Transit and Timetabling

2.1 Introduction

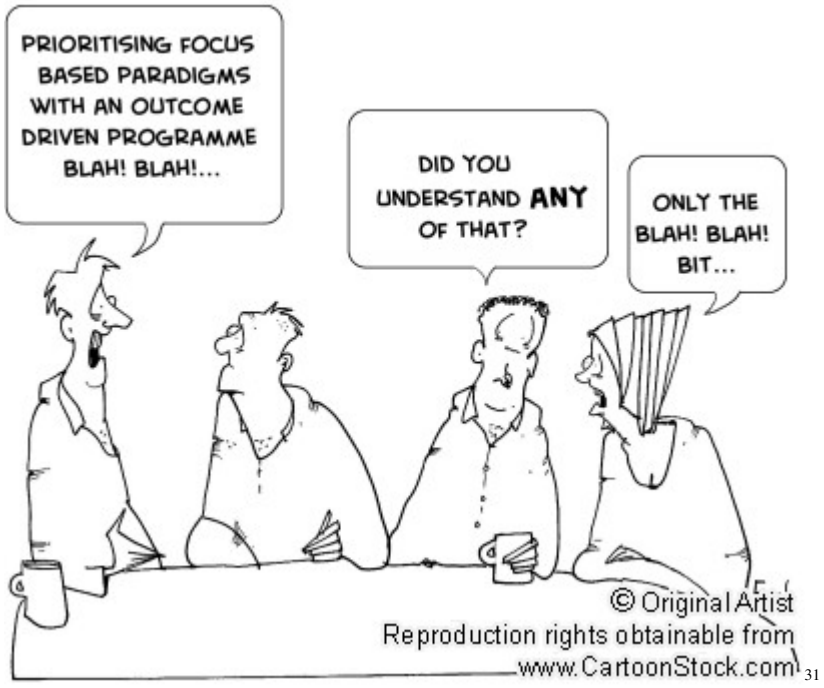
This thesis is about the timetabling of Bus Rapid Transit systems, and so to understand the work presented here, it is necessary to understand both BRT systems and the transit timetabling process. In this chapter both BRT systems and the transit timetabling process will be introduced, for readers unfamiliar with these areas. While introducing these topics, particular attention will be drawn, where necessary, to aspects of both BRT systems and the transit timetabling process that have a direct bearing on the task of investigating ultra-efficient BRT timetabling. In short, the purpose of this chapter will be to provide all the background information necessary to be able to clearly understand the research to be presented in the remainder of the thesis.

The first part of this chapter introduces BRT systems, looking at both the TransMilenio BRT, the test site for this thesis, and other BRT systems from around the world. Some definitions of BRT from the literature will be presented and discussed, the key characteristics of BRT systems will be examined, and a characteristics-based definition of BRT will be developed. With BRT systems in general introduced, some out of the ordinary characteristics of the TransMilenio BRT system that have a bearing on the work to be presented in this thesis will be discussed.

The transit timetabling process will then be introduced. As it is difficult to understand the transit timetabling process outside of the transit planning process — the entire data organisation process for

transit systems — the transit timetabling process will be presented in the context of the transit planning process. It will be argued that BRT timetabling, in particular high-load BRT timetabling, is a special case of timetabling, one far more complex than normal, everyday timetabling cases. In building this argument, parallels will be drawn between the special case of high-load BRT timetabling and the separate transit planning problem of organising vehicle schedules for normal, everyday transit network circumstances. Finally, the proposition will be raised that, as is the case with real-world vehicle schedules, more efficient high-load BRT timetables might be achievable by the application of highly computationally intensive solution approaches.

2.2 What is Bus Rapid Transit?



[31]

31 *Management Speak Cartoons*, CartoonStock Website, Available: http://www.cartoonstock.com/directory/m/management_speak.asp (Accessed 17 January 2009)

One would expect, in the literature, that posited answers to the question “What is Bus Rapid Transit?” would be grounded in the observation of transit systems that are widely agreed to be BRT systems, as opposed to normal bus systems or for that matter clearly distinct systems such as commuter railways. From such observations, the central characteristics of BRT systems might be discerned, and from these central characteristics a definition of what a BRT system is might be constructed. Although this method of moving from observation, to a list of characteristics, to a definition can be discerned in the literature, sometimes quite clearly, at other times there seems little connection between actual BRT systems and the definitions that purport to describe them.

The primary reason for this disconnect between some BRT definitions, and actual BRT systems, appears to be that the definitions have become suffused with the type of management language parodied in the cartoon at the start of this section. *Death Sentences: How Cliches, Weasel Words and Management-Speak Are Strangling Public Language,* a recent work on the penetration of management language into the language of other discourses, sounds a warning bell about this trend.[32] *Death Sentences* notes that “management concerns are relatively narrow — relative, that is, to life, knowledge and possibility. This alone makes marketing and managerial language less than ideal for a democracy *or a college*. In addition their language lacks almost everything needed to put into words an opinion or an emotion; *to explain the complex, paradoxical*

32 Watson, D. (2003), *Death Sentences: How Cliches, Weasel Words and Management-Speak Are Strangling Public Language*, Random House Australia, Sydney, p. 2

or uncertain . . . [emphasis mine]"[33] In a similar vein, the Plain English Campaign argues that it would be better for people to simply say "children need good schools if they are to learn properly" rather than emit sentences like "high-qualitys learning environments are a necessary precondition for facilitation and enhancement of the ongoing learning process."[34] A stream of BRT definitions suffuse with such language which came from fringe sources might be one thing, but many of the most notable examples are from the most authoritative sources.

For example, Mr Lloyd Wright, one of the two editors of the *Bus Rapid Transit Planning Guide,*[35] an 800 page tome that is arguably the key BRT text, says that "Bus Rapid Transit is high-quality, customer-oriented transit that delivers fast, comfortable, and low-cost urban mobility. It is not business as usual."[36] BRT itself may indeed not be business as usual, but it would appear that definitions of BRT are to be business language as usual. The use of the phrase "Bus Rapid Transit is" in the above quote suggests a definition is on its way, but this definition reads more like a mission statement than anything else. "High-quality, customer-oriented transit that delivers fast, comfortable, and low-cost urban mobility" certainly sounds like a good thing, but defines nothing about how such mobility is to be achieved; if it were not for the fact that the subject of the definition is Bus Rapid Transit, one would not even know if buses were to be used. Immediately after Mr Wright's definition

33 op. sit., pp. 2-3

34 *Before and After*, Plain English Campaign Website, Available: http://www.plainenglish.co.uk/examples/before_and_after.html (Accessed 11 Febuary 2009)

35 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York

36 Wright, L. (2003), *Bus Rapid Transit - A Global Review*, Institute for Transportation & Development Policy, New York, slide 3

of BRT, he provides a list of BRT characteristics, some of which, such as the use of “segregated busways”, contribute towards defining BRT, and others of which, such as “excellence in customer service”, contribute towards continuing the marketing language theme.[37]

A further example of a BRT definition which is grammatically structured as a definition, but that seems to define little, comes from no less a source than the Transit Cooperative Research Program, “a cooperative effort of the Federal Transit Administration, the Transportation Research Board, the Transit Development Corporation, and the American Public Transportation Association.”[38] In their report entitled *Bus Rapid Transit, Volume 1: Case Studies in Bus Rapid Transit* it is said that “BRT is a flexible, rubber-tired rapid transit mode that combines stations, vehicles, services, running ways, and ITS elements into an integrated system with a strong positive identity that evokes a unique image.”[39]

Other than the totally clear identification of BRT as a rubber-tired transit mode, this ‘definition’ does little more than raise questions. BRT “is a flexible, rubber-tired rapid transit mode”; flexible in what way, rapid compared to what? BRT “combines stations, vehicles, services, running ways, and ITS elements into an integrated system”; so do properly run railways. As for the question of BRT systems having “a strong positive identity that evokes a unique image” can this really be a defining aspect

37 ibid.

38 *Research, Technical Assistance & Training*, Federal Transit Administration Website, Available: http://www.fta.dot.gov/assistance/research_resources.html (Accessed 9 March 2009)

39 Levinson, H., Zimmerman, S., et al. (2003), *Transit Cooperative Research Program Report No. 90: Bus Rapid Transit. Volume 1: Case Studies in Bus Rapid Transit*, Transportation Research Board, Washington, D.C., U.S.A, p. 1

of BRT systems? What if one were to consider a bus-based transit system that operated on bus-only roadways, and that had enhanced station environments, level boarding and pre-boarding ticketing but that didn't have "a strong positive identity" or "evoke a unique image." Does such a system, exhibiting all the classical characteristics of BRT systems, cease to be a BRT system because it does not have "a strong positive identity" or "evoke a unique image"? Surely such a system is merely a badly marketed BRT system. Saying that BRT systems *should* have "a strong positive identity" and "evoke a unique image" is one thing, but to allow such operating recommendations to slip over into a system definition tends to lead to definitions that read more like wine reviews than technology descriptions; BRT has a precocious fruity bouquet and a clear crisp finish.

One reason for this drift towards management language, language unable to "explain the complex, paradoxical or uncertain",[40] might be that the question of what a BRT system actually is, is not an area free of contention. For example, Dr. Dario Hidalgo, Deputy General Manager of the TransMilenio BRT from June 2000 to September 2003,[41] saw a need for an article entitled *Towards a Better BRT Taxonomy* to help "clear up confusion regarding what is — and what is not — a true BRT system."[42]

With authoritative sources producing BRT definitions of uncertain utility, and with the question of what a BRT system actually is being sufficiently unclear so as to produce articles about "BRT Taxonomy", a

40 Watson, D. (2003), *Death Sentences: How Cliches, Weasel Words and Management-Speak Are Strangling Public Language*, Random House Australia, Sydney, p. 2

41 *Q&A With Darío Hidalgo: by Erico Guizzo* (2009), IEEE Spectrum Website, Available: http://www.itdp.org/STe/ste19/brt.html (Accessed 17 January 2009)

42 *Towards a Better BRT Taxonomy: by Dario Hidalgo* (2009), The City Fix Website, Available: http://thecityfix.com/towards-a-better-brt-taxonomy/ (Accessed 17 January 2009)

quick and simple answer to the question of “What is Bus Rapid Transit?” would appear to be elusive. Given this circumstance, looking for an answer to this question of “What is Bus Rapid Transit?” via an examination of the key characteristics of BRT systems, rather than via a definition of BRT, might prove more fruitful. Arguably the most authoritative source in the transportation discipline, the canonical *Urban Transit Systems and Technology* by Professor Vukan Vuchic,[43] provides such a list of key BRT features. *Urban Transit Systems and Technology* states that “the minimum features a bus line must have to be considered BRT are:

- Most of ROW [right-of-way] is category B [partially separated, but with crossings at grade] and limited sections of ROW category C [common streets with general traffic].
- Clearly designated stops/stations with passenger amenities spaced 300 to 500 metres apart.
- Regular or articulated buses with distinct appearance, good riding comfort, low floor or high platform, and multiple doors for easy and fast boarding/alighting at stops/stations.
- Services offered with regular headways throughout the day. - i.e., it is regular transit rather than commuter transit.
- Movement of buses along the line, dispatching at stops and passenger information are well organised and controlled by various ITS measures, guaranteeing reasonably high reliability.”[44]

43 Vuchic, V. R. (2007), *Urban Transit Systems and Technology*, John Wiley & Sons, New Jersey

44 op. sit., p. 69

Although this list of characteristics goes far further in clearly identifying what BRT is compared to the previously discussed definitions, and is free of management language, parts of the above list of minimum BRT characteristics raise similar questions as before. For example, take a hypothetical BRT system that meets all of the above list of minimum characteristics except that it had buses without a "distinct appearance"; would that not just be a badly branded BRT system? Similarly, a hypothetical BRT system that lacks only "good riding comfort"; would that not just be an uncomfortable BRT system? More seriously, stating that having "stops/stations . . . spaced 300 to 500 metres apart" is a "minimum feature" of BRT appears to be at best highly contestable. A quite comprehensive global survey of BRT systems in the *Bus Rapid Transit Planning Gui*de has 24 out of 38 systems, over 60%, listed as having an average stop spacing of more than 500 metres, and the average stop spacing for the systems in total is 857 metres."[45]

Having arguably the most authoritative general transportation text conflicting with arguably the key BRT text, over a matter as basic as stop spacing, throws doubt on simply accepting the above minimum feature list as a definitive answer to the question "What is Bus Rapid Transit?" As an alternative, a somewhat revised list of key BRT characteristics will now be presented, based substantially on Professor Vuchic's minimum features list above, but using as its evidentiary base real-world BRT feature data from the *Bus Rapid Transit Planning Gui*de.

The *Bus Rapid Transit Planning Guide* contains a comprehensive survey of the characteristics of 44 BRT systems from around the globe,

45 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, pp. 767-778

with the qualitative and quantitative characteristics of these systems presented in two separate sets of tables. The set of qualitative tables compares characteristics such as whether or not the BRT systems have an “independently operated and managed fare collection system” whereas the set of quantitative tables compares characteristics such as the “average dwell time at stations”, reported in seconds.[46] Characteristics from both the qualitative and quantitative sets of tables will be used to provide statistical support for this alternative key characteristics list. Key characteristics will be illustrated using photographs of individual BRT systems, as appropriate. The presentation of an alternative key features list will begin with a brief introduction to the TransMilenio BRT system, the test site for this thesis, so as to provide a general overview of one system, before looking at the characteristics of many systems.

Bogotá’s TransMilenio BRT system is a modern, high-load BRT system, with the system having been opened in 2000 and with it being listed as having 1,450,000 passenger trips per day in the 2007 *Bus Rapid Transit Planning Guide*.[47] Of the 37 BRT systems listed in the *Bus Rapid Transit Planning Guide,* for which passenger trips per day data is available, the TransMilenio has the second highest number of passenger trips per day, behind São Paulo’s Interligado BRT System.[48] In terms of peak ridership the TransMilenio has the highest level of 45,000 passengers per hour per direction (pphpd).[49] To give some idea of the scale of this peak ridership level, London Underground subway lines have

46 op. cit., pp. 754-778

47 op. cit., p. 767

48 op. cit., pp. 767-778

49 op. cit., p. 767

a peak capacity of the order of 35,000 pphpd.[50] The TransMilenio is able to service such a high peak ridership both by using the larger articulated type buses and by running these buses at very close headways, both of which characteristics can be seen in Plate 2.1 below.

Plate 2.1: Four Buses on the TransMilenio BRT System[51]

As can be seen in Plate 2.1, the TransMilenio uses a homogeneous fleet of 18 metre long articulated buses; larger bus sizes than this are available, for example the Curitiba BRT system in Brazil uses 25 metre long bi-articulated buses,[52] but 18 metre long single-articulated buses are

50 Quintain Estates and Development Plc (2004), *Delivering a New Wembley: Volume 1: Response Report incorporating Further Information*, Quintain Estates and Development Plc, London, p. 26

51 photo_transmilenio_danielsson.jpg, Winnipeg Rapid Transit Website, Available: http://www.winnipegrapidtransit.ca/Images/photo_transmilenio_danielsson.jpg (Accessed 8 July 2008)

52 Demery, L. W. Jr. (2002), *Bus Rapid Transit in Curitiba, Brazil - An Information Summary*, Publictransit.us (formally Carquinez Associates), Vallejo, California, p. 13

the largest buses that are readily available from a wide range of bus manufacturers. Each of the TransMilenio's buses is rated as being able to carry 160 passengers.[53] This is a very high passenger loading for buses of such size, and is made possible in part by the fact that the buses have very little seated capacity, with only 48 seats per bus.[54] Also, during busy times, when the buses are at or near their 160 passenger capacity, it was the author's experience[55] that the buses could fairly be described as crush loaded. It is noted by the article *Applicability of Bogotá's TransMilenio BRT System to the United States*, that due to "the upper crowding limits that transit users in North America are willing to accept" the normal position is that "a 60ft articulated (high floor) bus has a capacity of 100 to 120 people."[56]

Although large sized buses are a common characteristic of BRT systems they cannot be said to be a defining characteristic; an appreciable number of the BRT systems listed in the *Bus Rapid Transit Planning Guide* use standard non-articulated buses.[57] With regards to the vehicles used by BRT systems, the only defining characteristic that can be discerned is the completely obvious one, **that BRT systems use bus vehicles**, as opposed to using other vehicles, such as trains.

---

53 Cain, A. (Principal Investigator) 2006, *Applicability of Bogotá's TransMilenio BRT System to the United States - Final Report May 2006*, Federal Transport Administration, Department of Transportation, United States of America, Washington, D.C., U.S.A, p. viii

54 ibid.

55 In support of the research being conducted for this thesis a one week site visit was made to the TransMilenio BRT system during June 2006. During this site visit the author travelled on the TransMilenio BRT system for between 20 and 30 hours.

56 Cain, A. (Principal Investigator) (2006), *Applicability of Bogotá's TransMilenio BRT System to the United States - Final Report May 2006*, Federal Transport Administration, Department of Transportation, United States of America, Washington, D.C., U.S.A., p. 34

57 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, pp. 767-778

Of the four buses visible in Plate 2.1, three are approaching and one is moving away. If only the three approaching buses are considered, one is not stopping at the stop, one is stopping at the stop, and for one, the bus furthest away, it is not yet possible to tell if it will be stopping at the stop or not. Having three buses that are travelling in the same direction in view at once is not a rare occurrence on the TransMilenio system, indeed higher bus densities are regularly observable. With the TransMilenio's vehicle capacity 'only' being 160 passengers, and the peak loading being the previously mentioned 45,000 pphpd, very high bus densities are a necessity. Dividing this 45,000 pphpd peak passenger load by the bus capacity of 160 passengers, gives a peak bus rate of approximately 280 buses per hour, or one bus approximately every 13 seconds.

The theoretical potential for BRT systems to service very high passenger loads can be demonstrated by taking the number of buses per hour a roadway can service, available in works such as the canonical *Highway Capacity Manual*,[58] and multiplying that number by the passenger capacity of buses. Having actual systems operating at loads such as 45,000 passengers per hour per direction simply confirms the theoretical expectations. That said, although BRT is capable of servicing high passenger loads, actually servicing high passenger loads cannot be said to be a defining characteristic of BRT; there are BRT systems that service comparatively modest passenger loads. For example, the three lowest capacity BRT systems, in terms of peak passenger loads, in the *Bus Rapid Transit Planning Guide*, the ExM system in Eugene, Oregon,

58 Transport Research Board (2000), *Highway Capacity Manual 2000 (Metric Units)*, National Research Council, Washington, D.C., U.S.A

the Hangzhou City system in Hangzhou and the South Busway in Pittsburgh, have peak passengers per hour per direction levels of 500, 1,500 and 1,650 respectively.[59] In terms of defining characteristics of BRT, with regards to passenger capacity, it can only be said **that BRT systems are capable of servicing high passenger loads**.

A further key characteristic that can be observed with BRT systems is the type of roadway on which they operate. Previously, Plate 2.1 showed the TransMilenio BRT system operating on a bus-only roadway, in the middle of a normal roadway heavily loaded with traffic. Below, Plate 2.2 shows a section of the Curitiba BRT system also running on a bus-only roadway.

Plate 2.2: A Bi-articulated Bus on the Curitiba BRT System[60]

59 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, pp. 767-778

60 *Fares & Ticketing Systems*, citytransport.info Website, Available: http://www.citytransport.info/Fares.htm (Accessed 8 July 2008)

In Plate 2.2 one of the 25 metre long bi-articulated buses that provides the Curitiba BRT service can be seen running on a bus-only roadway that is not part of a normal roadway carrying general traffic. It is common-place for BRT systems to run on bus-only roadways, as has been shown for both the TransMilenio and Curitiba BRT systems. Of the 44 BRT systems listed in the *Bus Rapid Transit Planning Guide,* 32 are listed as operating on "segregated busways or bus-only roadways" and the remaining 12 are listed as partially operating on segregated busways or bus-only roadways.[61] None of the 44 BRT systems are listed as not running, at least partially, on a segregated busway or bus-only roadway.[62] So, with regards to rights-of-way, it can be said **that BRT systems at least partially operate on bus-only roadways**.

Another infrastructure related issue that distinguishes BRT is the use of 'proper' stops or stations, something more than just a shelter against inclement weather.

61 op. cit., pp. 767-778

62 ibid.

Plate 2.3: Hilyard Stop on the EmX BRT System[63]

Plate 2.3 above shows the Hilyard stop on the EmX BRT system in Eugene, Oregon. By comparison to the previously shown TransMilenio and Curitiba BRT systems, the EmX is a very low passenger load BRT system, with a peak ridership of only 500 pphpd, compared to 45,000 and 20,000 pphpd respectively for the TransMilenio and Curitiba systems.[64] Nevertheless, the minimum set of amenities, seating, rubbish bins and a system map, are present and overall the stop shows a greater resemblance to a light-weight railway station than it does to a bus shelter. As passenger load increases, BRT stops tend to more-and-more resemble full-amenity, full-size heavy-rail railway stations. At 20,000

63 *Hilyard Station on Flickr*, Flickr Website, Available: http://flickr.com/photos/functoruser/348559777/ (Accessed 19 July 2008)

64 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, pp. 767-768, 777

pphpd, the Curitiba BRT system is in the Rapid Rail Transit load range as specified by *Urban Transit Systems and Technology*,[65] and, as shown in Plate 2.2 its stops, although of an innovative prefabricated tubular construction, are also substantial constructions, far more than mere bus shelters. At 45,000 pphpd, the TransMilenio BRT system is well-and-truly in the Rapid Rail Transit load range, and, as shown in Plate 2.1, its stops look basically the same as heavy-rail railway stations.

Of the 44 BRT systems listed in the *Bus Rapid Transit Planning Guide*, 25 are listed as having an "enhanced station environment (i.e., not just a bus shelter)", 17 are listed as partially having an enhanced station environment and 2 are listed as not having an enhanced station environment.[66] So, with regards to stops, it could be said **that BRT systems very often have at least partially enhanced station environments**.

A further aspect of BRT systems is the use of level boarding. Plate 2.4 below shows passengers boarding a bus on the Optibus BRT system in León, Mexico.

65 Vuchic, V. R. (2007), *Urban Transit Systems and Technology*, John Wiley & Sons, New Jersey, p. 76

66 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, pp. 754-766

Plate 2.4: Level Boarding on the Optibus BRT system[67]

As can be seen above, the BRT stop platform height and the bus floor height match with sufficient precision to allow wheelchair users, or other users with mobility issues, such as the gentleman shown above on crutches, ready access.

67 *Access Exchange International – Photo Tour*, Access Exchange International Website, Available: http://www.globalride-sf.org/phtos.html (Accessed 12 July 2008)

Plate 2.5: An External Shot of an Optibus BRT Stop[68]

In Plate 2.5, an external shot of the same Optibus BRT system shown in Plate 2.4, it can be seen that level boarding on BRT systems is achieved in the same manner as for train systems, namely that the stop platform level is raised above the height of the road-bed. To match the stop platform level, the bus doors are also raised, as can be seen on the below plate of the boarding side of a TransMilenio bus.

68 ibid.

Plate 2.6: Boarding Side of a TransMilenio Bus[69]

Of the 44 BRT systems listed in the *Bus Rapid Transit Planning Guide,* 17 are listed as having “at-level boarding and alighting”, nine are listed as having partial at-level boarding and alighting and the remaining 18 are listed as not having at-level boarding and alighting.[70] With 18, or approximately 40%, of the listed BRT systems not having any degree of level boarding, it is not possible to say that level boarding is a defining characteristic of BRT. With several of the large ‘signature’ BRT systems, such as the TransMilenio in Bogotá and the Curitiba BRT, being entirely level boarding systems though,[71] it is possible to say **that BRT systems often feature level boarding**.

The final aspect of BRT systems that will be considered in this section is the use of pre-boarding ticketing.

69 *Maximizing the Air Quality Benefits of Bus Rapid Transit* (2004), Clean Air Initiative Website, Available: http://www.cleanairnet.org/baq2004/1527/articles-59335_hook.ppt (Accessed 3 March 2009), slide 30

70 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, pp. 754-766

71 op. cit., pp. 754-755

Plate 2.7: Ticket Gates on the TransJakarta BRT System[72]

In Plate 2.7 above, ticket gates on the Indonesian TransJakarta BRT system are shown. Pre-boarding ticketing is one of the ways in which Bus Rapid Transit attempts to meet the "rapid" part of its name. As is common-place with rail systems, many BRT systems handle the tasks of either purchasing or validating tickets as an entirely separate process that is completed before the passengers board their vehicles, thus improving boarding time and consequently service speed. Of the 42 BRT systems listed in the *Bus Rapid Transit Planning Guide* that are not fare-free, 15 are listed as having "pre-board fare collection and fare verification", 12 are listed as having partial pre-board fare collection and fare verification and the remaining 15 are listed as not having pre-board

72 *TransJakarta – Wikipedia*, Wikipedia Website, Available: http://en.wikipedia.org/wiki/TransJakarta (Accessed 12 July 2008)

fare collection and fare verification.[73] With 15, or approximately 35%, of the listed BRT systems not having any degree of pre-boarding ticketing, it is not possible to say that pre-boarding ticketing is a defining characteristic of BRT. Compared to the common-place on-board ticketing of normal bus systems, though, the regular use of pre-boarding ticketing on BRT is notable, and with approximately two-thirds of listed BRT systems having some degree of pre-boarding ticketing, it is possible to say **that BRT systems often feature pre-boarding ticketing**.

Six transit system characteristics have been examined in this section, namely vehicle type, load capability, right-of-way type, station type, boarding approach and ticketing approach. For each of these characteristics, the nature of BRT systems has been examined, with the assistance of the *Bus Rapid Transit Planning Guide*, and a short summary phrase given, highlighted in bold. Combining these short summary phrases together defines BRT as **a bus-based transit mode, capable of servicing high passenger loads, that at least partially operates on a bus-only roadway, that very often has at least partially enhanced station environments and that often features level boarding and pre-boarding ticketing.**

This definition could be converted into a classical (or archetypal) definition by removing the qualifying clauses, which would define a classical BRT system as **a bus-based transit mode, capable of servicing high passenger loads, that operates on a bus-only roadway, and that has enhanced station environments, level boarding and pre-boarding ticketing.** The TransMilenio BRT, the case

73 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, pp. 754-766

study site for this thesis, fully satisfies the requirements of this 'classical' BRT definition.

These above two characteristics-based definitions of Bus Rapid Transit, one general and one classical, are both free from management language and from, it would seem, highly contestable claims. However precise these two definitions may be though, they do not really capture the essence of what BRT is: they do not say *why* BRT has the above list of characteristics. One definition that tries to say why BRT has the characteristics it does is given by the Canadian Urban Transit Association which in one paper defines BRT as "a rubber tired rapid transit system with running ways, stations, and all the other attributes normally associated with rail rapid transit."[74] This definition is short, clear and indirectly provides the identifying characteristics of BRT by reference to railway systems.

The parallels between Bus Rapid Transit and Rail Rapid Transit are of particular note in the case of the TransMilenio BRT system. A Federal Transport Administration report describes a situation where there "were a total of 10 attempts to implement a heavy rail solution in Bogotá between 1947 and 1997. Over these years, Bogotá became the battleground between two opposing transit planning philosophies. First, the school that believed that a metro network ought to be the backbone of the transit system, complemented by bus-based services. Second, the school that believed that exclusive busways and Bus Rapid Transit style services could provide a similar level of service without the need for

74 McCormick Rankin Corporation (2004), *Bus Rapid Transit: A Canadian Industry Perspective*, Canadian Urban Transit Association, Toronto , p. 15

heavy rail."[75] Finally, "in 1997 Bogotá's mayor, Enrique Peñalosa, proposed a bus-based mass transit system modelled on those in operation in the Brazilian cities of São Paulo and Curitiba."[76] Enrique Peñalosa "wanted a reliable and free-moving bus system that was affordable and used road space at ground level. An underground or metro, he reasoned, was simply too expensive for a poor country . . . "[77]

Given this history, it is not surprising that the TransMilenio BRT system feels like a Rail Rapid Transit system right up to the moment when the station doors open and there is a bus standing there rather than a train. Cataloguing the characteristics of BRT systems is one way of defining BRT, but it fails to capture the conceptual genesis of BRT systems. A better way of answering the question "What is Bus Rapid Transit?" might simply be to say that Bus Rapid Transit is Rail Rapid Transit implemented using buses.

This section has addressed the first research question of this thesis, namely whether there is a clear definition of what is, and what is not, a BRT system in the literature. This question has been answered in the negative, with, as has been shown in this section, authoritative sources giving conflicting, unclear or highly contestable definitions of BRT.

---

75 Cain, A. (Principal Investigator) (2006), Appli*cability of Bogotá's TransMilenio BRT System to the United States - Final Report May 2006*, Federal Transport Administration, Department of Transportation, United States of America, Washington, D.C., U.S.A., p. 53

76 *Q&A With Darío Hidalgo:* by Erico Guizzo (2009), IEEE Spectrum Website, Available: http://www.itdp.org/STe/ste19/brt.html (Accessed 17 January 2009)

77 *The Message from Bogotá* (2003), Bristol Cycling Campaign Website, Available: http://www.bristolcyclingcampaign.org.uk/tbc/2003/summer/bogota.htm (Accessed 17 January 2009)

2.3 Bogotá's TransMilenio Bus Rapid Transit System

In the previous section the general characteristics of BRT systems were reviewed by examining a range of BRT systems from around the world. The case site for this thesis though is a single BRT system, the TransMilenio BRT system in Bogotá, Colombia. The TransMilenio meets the definition of a 'classical' BRT system as given in the last section, having characteristics including the use of a bus-only roadway and pre-boarding ticketing. The TransMilenio also has some non-standard characteristics that affect the ultra-efficient timetable generation process, and these will be covered in this section. Specifically there are three features of the TransMilenio system that affect timetabling, namely the preponderance of non-pre-empted stop lights, the presence of overtaking lanes at stops and the fact that the TransMilenio is not only capable of servicing high loads but that it actually does service high loads.

As is a common occurrence with transit systems, the TransMilenio BRT system was retrofitted to an existing, established city and so had to, at least to some degree, harmonise with the existing roadway system.[78] Indeed, the construction of the TransMilenio BRT began in 1999,[79] by which time Bogotá was already a significant metropolis with a population approaching seven million people,[80] and with a well established arterial

78 *TransMilenio Busway-Based Mass Transit, Bogotá, Colombia*, The World Bank Website, Available: http://siteresources.worldbank.org/INTURBANTRANSPORT/Resources/Factsheet-TransMilenio.pdf (Accessed 17 January 2009), p. 14

79 *TransMilenio's Contributions to the Development of Bus Rapid Transit Systems: by Darío Hidalgo Guerrero*, Ph.D., Bogotá Lab Website, Available: http://www.bogotalab.com/articles/transmilenio.htm (Accessed 26 May 2008)

80 *Censo General 2005 - Resultados Área Metropolitana de Bogotá*, Departamento Administrativo Nacional de Estadistica Website, Available: http://www.dane.gov.co/files/censo2005/resultados_am_municipios.pdf (Accessed 17 January 2009). p. 3

road system and a large number of traffic lights. The BRT Planning Guide notes that “most BRT systems to date have been developed in developing countries with high bus frequencies and relatively few intersections” and so “have not relied heavily on sophisticated real time signalling systems.”[81] In common with most BRT systems, the TransMilenio does not use traffic light pre-emption;[82] in distinction to most BRT systems the TransMilenio lines encounter a great many traffic lights. The high frequency of traffic lights has meant that the impact of these traffic lights on service speed has needed to be treated as a central, rather than peripheral, aspect of the timetable development process.

A further special characteristic of the TransMilenio that affects the timetable development process is that the system has passing lanes at its stops, which, as this allows one bus to pass another, allows for express bus services to be run. Indeed, the relationship between the presence of passing lanes at stops and the running of express services is sufficiently close that the *Bus Rapid Transit Planning Guide* lists “overtaking lanes at stations/provision of express services” as a single characteristic.[83] Passing lanes are not a common characteristic of BRT systems. Of the 44 BRT systems listed in the *Bus Rapid Transit Planning Guide,* the TransMilenio is one of only five systems listed as having “overtaking lanes at stations”, with 7 being listed as partially having overtaking lanes at stations and 32 being listed as not having overtaking lanes at stations.[84] As will be

81 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, p. 314

82 From notes taken at a meeting between the author and Alejandro Niño of the TransMilenio S.A. Planning Department on the 7th of June 2006.

83 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, pp. 754-766

84 ibid.

discussed in some detail later, the TransMilenio system uses express buses extensively to improve the speed and efficiency of its operations. Consequently, for an ultra-efficient timetable to be able to be meaningfully compared to the existing TransMilenio timetable such a timetable needs to be constructed with express services where they are beneficial. How this special characteristic of TransMilenio, being able to run express services, specifically affects the development of an ultra-efficient timetable will be discussed at length in later parts of this thesis.

The final special characteristic of the TransMilenio BRT system that affects the modelling, is that the TransMilenio is not just capable of servicing high loads but in fact does service high loads. Previously, the high-load capability of BRT systems has been covered, but capabilities do not have to be utilised and there are numerous BRT systems servicing low load levels. The TransMilenio not only services high passenger loads, but as measured in terms of passengers per hour per direction, it is the highest load BRT system in the world, servicing a peak load of 45,000 pphpd.[85]

This section has identified three special characteristics of the TransMilenio BRT system that affect the construction of an ultra-efficient timetable; namely the high frequency of traffic lights, the presence of passing lanes at stops and the servicing of high passenger loads. Specifically how these three special characteristic of the TransMilenio system will be accounted for in the development of an ultra-efficient timetable is a subject that will be covered at some length in later parts of this thesis.

85 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York. p. 767

2.4 An Overview of Transit Timetabling and Scheduling

This thesis is about the timetabling of BRT systems, with this chapter being dedicated to introducing both timetabling and BRT systems. Having now introduced BRT systems in general, and the TransMilenio system in particular, this section will begin the process of introducing timetabling. Timetabling is, though, only one part of the task of planning the operation of a transit system; the scheduling of driver crews being an example of another part of the transit planning task. To clearly explain transit timetabling in context, the whole of the transit planning task will be reviewed in this section. As one of the arguments to be built in later sections of this chapter is that there is an interesting parallel between vehicle scheduling and the ultra-efficient timetabling being explored in this thesis, the vehicle scheduling aspect of transit planning will be covered in some detail.

Professor Avishai Ceder, one of the 'pioneers' in the field of transit scheduling,[86] says that the "bus, railway and passenger ferry operational planning process includes four basic components performed usually in sequence:

1. Network route design
2. Setting timetables
3. Scheduling vehicles to trips
4. Assignment of drivers (crew)"[87]

86 Ceder, A., Stern, H. I. (1981), *Deficit Function Bus Scheduling with Deadheading Trip Insertions for Fleet Size Reduction*, Transportation Science, Volume 15, Number 4, November 1981

87 Ceder, A. (2003), *Public Transport Timetabling and Vehicle Scheduling*, In: Lam, W.H.K., Bell, G.H., *Advanced Modeling for Transit Operations and Service Planning*, Elsevier Science Ltd, Pergamon, p. 31

Although the above four steps are largely self-explanatory, as one of the objectives of this chapter is to draw a parallel between timetabling on the one hand and vehicle scheduling on the other, there is a benefit to detailing what the above four listed tasks exactly refer to. The task of "network route design" consists of deciding exactly where a transit line is to be placed and exactly where each stop will be placed along that transit line. Once the topology of a line is known, it is possible to estimate expected passenger load on the line, which allows timetables to be set. "Setting timetables" is the process of calculating the times that transit vehicles will run at, and which stops they will stop at, with the frequency of the vehicle trips being sufficient to meet estimated passenger load. "Scheduling vehicles to trips" is the process of assigning a particular vehicle in a transit fleet to each of the timetabled trips and joining up the trips so that they begin and end at depots. In the same way that a vehicle is needed to service a trip in non-automated systems so is a driver, with "assignment of drivers (crew)" referring to the process of assigning a particular driver in a transit-driver-crew to each of the timetabled trips.

The above four steps show the informational aspects of transit planning as a directed sequence of steps, with little or no feedback from one step to previous steps. Professor Ceder, in introducing his overview of the transit planning process, states that although it would be "desirable for all the four components to be planned simultaneously to exploit the system's capability to the greatest extent and maximise the system's productivity and efficiency [the] planning process is extremely cumbersome and complex, and therefore [it] seems to require separate treatment of each component, with the outcome of one fed as an input to

the next component."[88] The transit timetabling process, the topic of this thesis and step two in the previous list, takes passenger load data, which can be estimated once the network route design has been completed, and the network route design itself, as input data. As output data, the transit timetabling process produces vehicle trip data, which is constituted of the times at which vehicles depart and, in systems that use expresses, the stopping patterns of the express vehicles.

The role of a transit timetable is to service passenger load, and an efficient timetable is one that does so using the minimum number of transit vehicle trips, or one that minimises some other specified operational characteristic. To determine the minimum number of transit vehicles required to service a given passenger load, it is necessary to know the passenger capacity of the vehicles, which is not an absolute as it may vary depending on legal passenger load limits, the layout of the interior of the vehicles and the minimum acceptable comfort level. Another common-place factor in the construction of an efficient timetable is a minimum service guarantee, which sets a lower boundary for how often transit vehicles can run. For example, a given transit line's operator might guarantee that the line will be serviced at least every 30 minutes while it is open. Together these data points, passenger load, load factor and a minimum service frequency guarantee, are sufficient to calculate the service frequency for a given time period using a calculation method known as the maximum load procedure.[89] The maximum load procedure states that the service frequency should be set to be the maximum passenger load divided by the passenger capacity of the vehicles, or the

88 ibid.

89 op. sit., p. 37

minimum service frequency, whichever is greater. In assessing the maximum passenger load it is important to use the maximum load point along the entire route so as to ensure against under-servicing of the line. Methods for determining the maximum load point are discussed in papers such as *Updating Ride Checks with Multiple Point Checks*.[90]

This remarkably straight-forward maximum load procedure calculation is used "across almost all the public transport agencies."[91] In transit planning, a very strong case could be made that calculating timetables is normally the simplest of the tasks that needs to be undertaken, and that, for timetabling professionals, the task of clearly communicating their timetables to the general public is where the real 'challenges' lie vis-a-vis timetabling. The task before timetabling, network route design, often involves the use of sophisticated geographical information systems to lay out the lines and complex statistical techniques to compute the expected passenger load levels. The two processes after timetabling, vehicle and driver scheduling, are of sufficient complexity that a small industry has developed to provide software packages to assist with the process, such as the Hastus software suite produced by the Giro company, based in Montreal, Canada.[92] Although most real-world timetabling circumstances represent a far simpler problem than the tasks before and after timetabling, not all transit timetabling is of the simple type. One example of a more complex timetabling situation is clock-face timetabling.

90 Furth, P. G. (1989), *Updating Ride Checks with Multiple Point Checks*, Transportation Research Record, Volume 1209

91 ibid.

92 *HASTUS: An Integrated Solution for Transit Scheduling and Operations* (2007), Giro Company Website, Available: http://www.giro.ca/docs/public/pdf/hastus/en/GIRO-MKT-HASTUS-PST-ARCHE-20070322.pdf (Accessed 7 February 2009)

Clock-face timetabling is a type of timetabling that attempts to align vehicle departure times at interchanges to easy to remember times, such as on the hour, half-hour or quarter-hour. By having transit vehicles all arrive just before a given ‘clock-face’ time, say just before every quarter hour, and then depart on the quarter hour, not only are the departure times memorable, but the wait time when transferring between transit vehicles is minimised. Achieving the synchronisation of transit vehicle departure times at interchanges though presents problems, as there is no reason that the travel time between interchanges should fall into convenient 15, 30 or 60 minute durations. The Swiss Federal Railways (SBB) have made notable use of clock-face timetabling. In a recent report to the Swiss Federal Railways, Professor Jeff Kenworthy notes that “as with the freight system, SBB’s passenger network is a very dense and interconnected one that relies heavily on transfers or connections between services to get from one’s origin to destination. To make this simpler and more reliable, SBB introduced a symmetrical integrated clock face timetable, meaning that services between centres leave and arrive on set rhythms of 30 minutes and 60 minutes at most stations”[93] and that under this system “2.11% [of connections] were missed.”[94] The Swiss clock-face timetabling system has garnered sufficient interest that a paper has been produced discussing the system’s possible applicability to North America, namely *Intercity Rail Fixed-Interval, Timed-Transfer,*

93 Kenworthy, J. (2007), *Making Connections: Growing Public Transport’s Market Share in Switzerland Through a More Customer and Information-Oriented Approach* (A Report to the Swiss Federal Railways), Murdoch University, Perth, p. 42

94 op. sit., p. 11

Multihub System: Applicability of the Integraler Taktfahrplan Strategy to North America.[95]

In other areas of timetabling, it can also be seen that 'special circumstances' are required before the timetabling problem becomes genuinely difficult. At the recent Computer-Aided Scheduling of Public Transport (CASPT) conference in 2006, all of the research focus with regards to timetabling was centred around unusual timetabling situations, or concerns beyond pure efficiency. For example, one paper, *Operations Control Strategies to Improve Transfers Between High-Frequency Urban Rail Lines* focused on the real-time selective delaying of timetabled departure times so as to improve transfers.[96] A further paper, *Delay Resistant Timetabling*, focused on the balance between the benefit of delay resistance and the cost of timetabling in extra 'slack' time so as to provide this delay resistance.[97]

The example of clock-face timetabling and the examples of timetabling research from the recent CASPT 2006 conference, demonstrate the degree to which constraints or complications have to be introduced to raise the complexity of the timetabling problem above the level that can be solved by the application of the previously discussed "maximum load procedure" formula. No such constraints or complications are needed to raise the problem of vehicle and driver

95 Maxwell, R. R. (1999), *Intercity Rail Fixed-Interval, Timed-Transfer, Multihub System: Applicability of the Integraler Taktfahrplan Strategy to North America*, Transportation Research Record, Volume 1691

96 Wong, C., Wilson, N. (2006), *Operations Control Strategies to Improve Transfers Between High-Frequency Urban Rail Lines*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds

97 Liebchen, C., Stiller, S. (2006), *Delay Resistant Timetabling*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds

scheduling above the level that can be solved by the application of a simple formula. As previously mentioned, a small software industry has developed to help operators address the problem of efficiently scheduling vehicles and drivers.

In conclusion, this section has reviewed the timetabling and scheduling process and its main components, namely network route design, setting timetables, scheduling vehicles to trips and assignment of drivers. Although the process of network route design, and predicting expected passenger load is an interesting one, involving at times the application of quite sophisticated techniques, network route design is not one of the key themes of this thesis, and so will not be considered further. Network route, the location of stops, expected passenger load, and all such aspects of this phase will, for the process of ultra-efficient timetable development simply be treated as ‘given’ inputs. Timetabling, the second phase in the transit data process described by Professor Ceder, is the primary theme of this thesis and will be further examined throughout this, and following, chapters. Vehicle and driver scheduling, like network route design, is not a key theme of this thesis, but as there is an interesting parallel to be drawn between present-day vehicle scheduling techniques and the experimental timetabling techniques being researched here, vehicle scheduling techniques will be considered further in the next section. In particular, the next section will address the question of why the general vehicle scheduling case is a complex problem.

2.5 Transit Scheduling and the Combinatorial ‘Explosion’

In the previous section a fundamental difference between transit timetabling and transit scheduling was discussed, namely that, in the general case, timetabling is a simple problem to solve whereas vehicle and crew scheduling is a complex problem to solve. The reason that transit timetabling is a simple problem was shown via a discussion of the fairly simple components of the problem, such as the maximum load point and the minimum service guarantee level. Furthermore, the means of solving the transit timetabling problem, the fairly straight-forward maximum load procedure formula was examined. This section will show, by contrast, why the transit scheduling problem is a complex problem and briefly examine the often highly sophisticated means used to solve this problem. The objective of discussing transit scheduling in such depth, when it is not a key theme of this thesis, is to allow for a clear parallel to be drawn between normal transit scheduling and the special case of high-load BRT timetabling. With this purpose in mind, this section will focus on the case of vehicle scheduling, rather than crew scheduling, because, as both timetabling and vehicle scheduling are concerned with vehicles, this allows for a more direct comparison.

To begin examining why the vehicle scheduling problem is a complex problem, it is necessary to know the exact parameters of what has, so far, somewhat vaguely been referred to as simply the vehicle scheduling problem. A paper called *Progress in Solving Large Scale Multi-depot Multi-vehicle-type Bus Scheduling Problems with Integer Programming*[98]

98 *Progress in Solving Large Scale Multi-depot Multi-vehicle-type Bus Scheduling Problems with Integer Programming: by Uwe H. Suhl, Swantje Friedrich and Veronika Waue* (2007), Association for Information System Electronic Library Website, Available: http://aisel.aisnet.org/cgi/viewcontent.cgi?article=1080&context=wi2007 (Accessed 3 March 2009), p. 430

lays out the parameters of the vehicle scheduling problem very clearly, detailing what needs to happen to the group of timetabled trips generated at the timetabling step of the transit planning task. The paper states that “each timetabled trip can be served by a vehicle belonging to a given set of vehicle types. Each vehicle has to start and end its work day in one of the given depots. After serving one timetabled (loaded) trip, each bus can serve one of the trips starting later from the station where the vehicle is standing, or it can change its location by moving unloaded to an another station (deadhead trip) in order to serve the next loaded trip starting there.”[99] Although this process sounds relatively straight-forward, the ‘freedom of choice’ of the buses evident in the above description, is central to understanding why transit scheduling is a complex problem. This ‘freedom of choice’ can be made clearer by amplifying the language of part of the above description somewhat, as follows: “After serving one timetabled (loaded) trip, each bus can serve [any] one of the trips starting later from the station where the vehicle is standing, or it can change its location by moving unloaded to [any other] station (deadhead trip) in order to serve the next loaded trip starting there.”

So, after each trip a bus may be used to service a number of other trips, either at the station it is at, or by moving empty to other stations and then servicing a trip leaving from there. Selection of any particular trip, will change the time and location that the bus is next free and so will change the next group of trips that the bus may be used to service. As each selection of a trip changes the next set of trips that a bus can be used to service, the number of different combinations of trips can end up

99 op. sit., p. 430

being very large. Take a simple example where for a given bus there are 15 transitions between trips and three choices at each transition. The number of possible trip combinations is therefore three to the power of 15, or approximately 14 million (14,348,907) combinations. Because of the nature of the calculation, raising one number to the power of another, the number of combinations goes up very quickly as either number is increased. For example, if a bus had the same 15 transitions between trips but six, rather than three choices at each transition, then the number of possible combinations would be approximately 500 billion (470,184,984,576).

Whether it be 14 million or 500 billion combinations of trips to choose from, for a given method of judging efficiency, one of these combinations will be the most efficient, the optimal, solution. A paper called *Dynamically Configured λ-opt Heuristics for Bus Scheduling,* discusses judging efficiency in terms of minimising the "operational cost and the number of buses" stating that "bus scheduling is a complex combinatorial optimisation problem. The operations planning and scheduling process starts off with the design of a timetable of trips that have to be taken by buses. Each trip has a starting time/location and a destination time/location. Bus scheduling involves assigning a set of trips to a set of buses such that:

1. The sequence of the trips for each bus is time feasible: no trip precedes an earlier trip in the sequence.
2. Each trip is served by exactly one bus.

There are two types of operational cost involved, these are layover (idle time) and dead running (relocating the bus between locations without

passengers, this includes leaving and returning to the depot, when a journey doesn't begin or end at a depot). The objective is to minimise the operational cost and the number of buses. Given the nature of the problem it is almost always impossible to reduce the layover and dead running cost to zero. The complexity of the problem quickly increases as the number of the depots and the types of vehicle increases."[100]

The title of the paper that the above quote comes from, *Dynamically Configured λ-opt Heuristics for Bus Scheduling,* gives an indication that the techniques used to solve the vehicle scheduling problem are somewhat more sophisticated than the "maximum load procedure" used to solve the transit timetabling problem. The reason for the difference is a direct result of the nature of the problem. In the case of transit timetabling there is not a vast array of combinations to choose from, merely a few operational numbers that need to be acquired which can then be processed via a relatively simple formula. With vehicle scheduling, the number of combinations that need to be assessed goes up very fast as the size and complexity of transit networks increase; a situation somewhat informally referred to as the combinatorial 'explosion.'

The techniques that have been developed to deal with this combinatorial 'explosion' in the domain of vehicle scheduling can be divided into two groups, manual and computerised. One early manual method for arriving at efficient vehicles schedules was called the deficit function. This method involved graphing "the deficit number of vehicles

100 Rattadilok, P., Kwan, R. S. K. (2006), *Dynamically Configured λ-opt Heuristics for Bus Scheduling*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds, p. 1

required at the particular terminal in question, in order to service the subset of trips servicing the terminal. The deficit function approach has its appeal in its visual simplicity and as a practical tool in the development of heuristics for handling more realistic scheduling problems."[101] A full description of the deficit function approach is given in the paper the preceding quote came from, *Deficit Function Bus Scheduling with Deadheading Trip Insertions for Fleet Size Reduction,* a paper from 1981.[102]

In more recent years, with what were once considered super computer levels of computing power becoming readily available, computerised approaches have become commonly used. For example, the HASTUS transit scheduling software package, produced by the Giro company mentioned in the last section, is used by "more than 250 transit companies in 22 countries"[103] and the Giro company is only one of a number of companies providing such software tools.

One way of applying computer power to 'solve' the vehicle scheduling problem, is to enumerate each possible set of trip combinations, assess each possibility for its efficiency, and select the most efficient option. Although this approach has much in its favour in terms of simplicity, and in the fact that it finds the optimal rather than a good solution, unfortunately it runs into problems in normal vehicle scheduling circumstances due to the impact of the previously discussed

101 Ceder, A., Stern, H. I. (1981), *Deficit Function Bus Scheduling with Deadheading Trip Insertions for Fleet Size Reduction*, Transportation Science, Volume 15, Number 4, November 1981, p. 341

102 ibid.

103 *HASTUS: Transit Scheduling and Operations* (2008), Giro Company Website, Available: http://www.giro.ca/docs/public/pdf/hastus/en/GIRO-MKT-HASTUS-PST-HASTUSE-20081124.pdf (Accessed 13 Febuary 2009)

combinatorial explosion. With the number of combinations that need to be assessed going up at an exponential rate, the computational work required to find the optimal solution can rapidly go beyond what even the most powerful computers can complete in any reasonable amount of time. When a combinatorial problem cannot be optimally solved in a reasonable amount of time, it is referred to as intractable.[104] The assessment of one paper, *Vehicle and Crew Scheduling for Urban Bus Lines*, published in 2006, was that even with the substantial amount of computer power readily available in recent times that, for multi-depot situations, "problems with 30 or more trips remained intractable",[105] which means that most transit networks are intractable, vis-a-vis vehicle scheduling. One provider of transit planning software, Omnibus, considers 30 vehicles a small operator, not 30 trips, stating in their promotional material that there "are over 150 sites using Omnibus solutions, both in the UK, and internationally. These clients range in size from small independent operators with under 30 vehicles to large companies within the major UK groups with in excess of a 9000 vehicles."[106]

Various different techniques[107] have been developed, in particular by the operations research discipline, to handle the fact that the vehicle

104 Hopcroft, J. E., Motwani, R., Ullman, J. D. (2006), *Introduction to Automata Theory, Languages, and Computation* (Third Edition), Addison-Wesley, Boston, pp. 425-482

105 Rodriguesa, M. M., de Souzab, C. C., Mourab, A. V. (2006), *Vehicle and Crew Scheduling for Urban Bus Lines*, European Journal of Operational Research, Volume 170, Issue 3, pp. 844-862

106 *Ominbus: Our Clients*, Omnibus Website, Available: http://www.omnibus.uk.com/clients.html (Accessed 13 Febuary 2009)

107 Bunte, S., Kliewer, N., Suhl L. (2006), *An Overview on Vehicle Scheduling Models in Public Transport*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds

scheduling problem is normally intractable, and to provide 'good' solutions in a reasonable amount of time. The research conducted in this thesis will, for reasons that were discussed in section 1.3, "Scope and Approach of the Research", not be making direct use of these sophisticated techniques, ones that go by names such as linear programming,[108] genetic algorithms[109] and simulated annealing.[110] Consequently, this section will not be detailing how such sophisticated techniques actually work; one does not dip ones toe into the waters of a domain of knowledge that produces papers with titles like *Bi-objective Evolutionary Heuristics for Bus Drivers Rostering*[111] unless one is planning to go for a really good swim.

In the context of the work to be done in this thesis, though, it is worth knowing that combinatorial problems do not become unsolvable when they become intractable; it is just that optimal solutions become unavailable. Good solutions are still available, via the application of computational techniques that, although highly sophisticated, are also well understood. Furthermore, these techniques have been successfully packaged into software by numerous companies and effectively applied to real-world transit systems by a wide range of transit operators. The

108 *Linear Programming: A Concise Introduction: by Thomas S. Ferguson* (2008), UCLA Department of Mathematics Website, Available: http://www.math.ucla.edu/~tom/LP.pdf (Accessed 22 January 2008)

109 Thangiah S. R., Osman I., Sun T. (1997), *Hybrid Genetic Algorithm, Simulated Annealing and Tabu Search Methods for Vehicle Routing Problems with Time Windows* (Working Paper), Institute of Mathematics and Statistics, University of Kent, UK

110 Souai, N., Teghem, J. (2006), *Hybridizing the Genetic Algorithm and the Simulated Annealing for the Airline Crew Rostering Problem*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds

111 Moz, M., Respício, A. and Vaz Pato, M. (2006), *Bi-objective Evolutionary Heuristics for Bus Drivers Rostering*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds

previously mentioned Giro company states that “in repeated tests and implementations all over the world, HASTUS scheduling algorithms produce savings ranging from 2 to 5% and more, over both competing systems and manual methods.”[112] The Giro company further states, as an example of its software’s effectiveness, that in “Montréal, the STM [Société de Transport de Montréal] achieves savings of $4M annually [and that] in Calgary an official city report quotes savings of $1.2M per year.”[113]

This section has outlined the scale of the complexity difference between normal timetabling problems and normal vehicle scheduling problems. Normal timetabling problems are not combinatorial problems, and can be efficiently solved, as was discussed in the last section, using the “maximum load procedure.” Normal vehicle scheduling problems are not only combinatorial problems but in normal-sized transit systems are intractable combinatorial problems, solved via the application of highly sophisticated computational techniques. The scale of the complexity difference between normal timetabling and normal scheduling problems is stark, as can be seen by the complexity of the tools that are used to solve these respective problems. For example, the Tracsis suite of transit scheduling tools[114] was described by one experienced scheduler[115] as being a “fearsome beast”, a description that could not be fairly applied to the

112 *HASTUS: Transit Scheduling and Operations* (2008), Giro Company Website, Available: http://www.giro.ca/docs/public/pdf/hastus/en/GIRO-MKT-HASTUS-PST-HASTUSE-20081124.pdf (Accessed 13 Febuary 2009)

113 *HASTUS: Transit Scheduling, Daily Operations and Customer Information* (Brochure) (2006), Giro Company, Montréal, p. 1

114 *Intelligent Crew Optimisation Scheduling Software*, Tracsis PLC Website, Available: http://www.tracsis.com/crew_scheduling_products.htm (Accessed 13 Febuary 2009)

115 A comment overheard by the author during informal discussions at The 10th International Conference on Computer-Aided Scheduling of Public Transport.

“maximum load procedure” formula used to solve normal timetabling problems.

2.6 High-load BRT Timetabling: A Special Case

In the previous section the differences between normal timetabling and normal vehicle scheduling were discussed, with normal vehicle scheduling being shown to be a far more complex problem than normal vehicle timetabling, because it is a combinatorial problem. In this section, it will be argued that high-load BRT timetabling is also a combinatorial problem, and so may benefit from the application of timetabling techniques that recognise its combinatorial nature, similarly to the way vehicle scheduling has benefited from such techniques. As such, it will be argued that the timetabling of high-load BRT systems should be viewed in a similar way to the previously discussed examples of clock-face timetabling or delay resistant timetabling; a special case of timetabling where the use of significantly more sophisticated timetabling techniques may well yield benefits.

High-load BRT systems become a special case of timetabling as a consequence of an aspect of their infrastructure. One of the approaches that designers of BRT systems use if they need to service high passenger loads, is the provision of passing lanes at stops, which allows one bus to pass/overtake another. If one bus can pass another at a stop, one or more express services can be run alongside an all-stop service, thus increasing overall service capacity. As discussed previously, while looking at the out-of-the-ordinary characteristics of the TransMilenio system, the provision of passing lanes is not a particularly common characteristic of BRT

systems. Of the 44 BRT systems listed in the *Bus Rapid Transit Planning Guide*, only five are listed as having “overtaking lanes at stations”, with seven being listed as partially having overtaking lanes at stations, leaving 32 systems that do not have overtaking lanes at stations.[116] Nevertheless, the highest load BRT system in the *Bus Rapid Transit Planning Guide* in terms of pphpd, the TransMilenio, has passing lanes, and the second highest load system, partially has passing lanes.[117] This relationship between overtaking lanes at stations and high-capacity BRT is mentioned by Professor Vuchic in *Urban Transit Systems and Technology* where, after discussing the general bus line capacity case of up to 90 buses an hour, he states that if “large stop areas permit bypassing of buses and dispatching supervision is provided, frequencies of 120 and exceptionally as high as 180 buses/hour [Portland (Oregon)] can be achieved.”[118]

Switching from an all-stop bus system to one with expresses, not only allows BRT systems to service very high passenger loads, it also fundamentally changes the nature of the timetabling problem, from a ‘simple’ one to a combinatorial one. The fundamentally combinatorial nature of express-service BRT timetabling is best demonstrated with the assistance of an illustration. Figure 2.1 below shows a conceptual diagram, produced by the Metropolitan Transportation Authority of the State of New York, of the services on their BRT system.

116 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, pp. 754-766

117 ibid.

118 Vuchic, V. R. (2007), *Urban Transit Systems and Technology*, John Wiley & Sons, New Jersey, p. 190

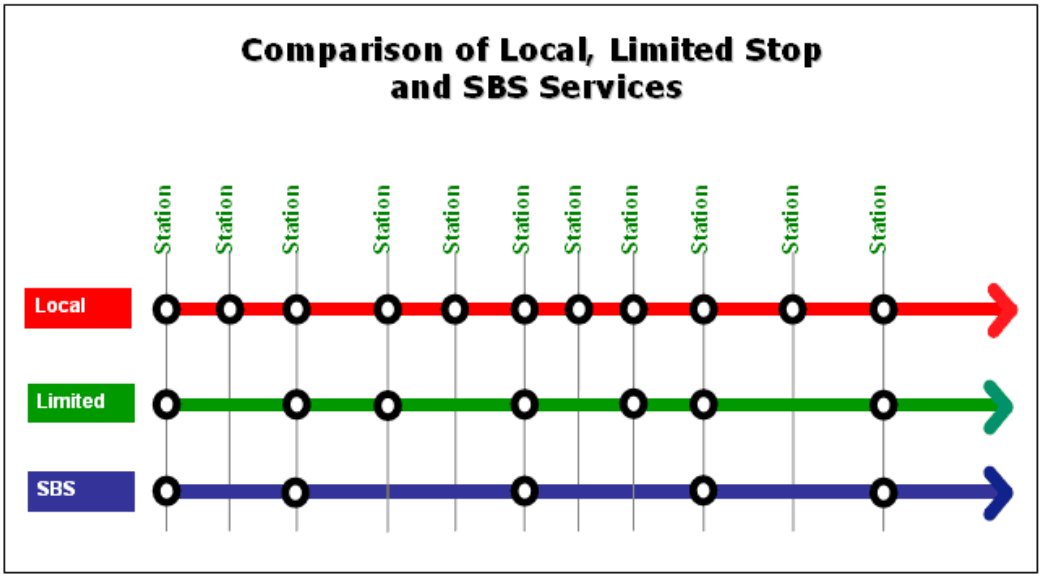


Figure 2.1: Conceptual Diagram of the Select Bus Service BRT System[119]

In the above diagram it can be seen that three different types of service, local, limited and SBS (Select Bus Service) are running along the same line, past the same set of stations/stops. The local service, stops at all the stops, the limited service stops at seven of the eleven stops that are shown, and the SBS service, stops at only five of the eleven stops. The exact length of the line above is not shown on the above conceptual diagram, with the arrows at the right hand side of the diagram indicating that such a line would be expected to be longer, with, presumably, eleven stops having been sufficient to make the point.

Using the above conceptual diagram as a basis from which to construct a simple concrete timetabling problem, the combinatorial nature of express-service BRT timetabling becomes readily apparent, a point that can be shown using only one express service, rather than the two express services shown in Figure 2.1 above. For example, consider a

119 *New York City Bus Rapid Transit Project*, Metropolitan Transportation Authority - State of New York Website, Available: http://www.mta.info/mta/planning/sbs/whatis.htm (Accessed 3 March 2009)

bus line of 27 stops, the average length of lines on the London Underground,[120] arguably the world's best known transit system. If you classify the first and last of these 27 stops as turn-around stops, that all buses must stop at, that leaves 25 intermediate stops, stops that the single express service of this example may or may not stop at. With two choices, to stop or not to stop, and 25 stops at which to make that choice, the number of possible stopping patterns is two raised to the power of 25, which gives approximately 34 million (33,554,432) possible stopping patterns.

So in the case of timetabling an ordinary length transit line with one express service, the problem is already looking very different in character to normal non-express timetabling. Rather than a simple "maximum load procedure" formula to apply, there are approximately 34 million options to choose from, one of which, given an appropriate metric, will be the best. If, as is the case in Figure 2.1 above, two express services are used, it should be noted that the number of stopping pattern options does not double, but goes up exponentially, to two to the power of 50, or 1,125,899,906,842,624 options. So a timetable with two express services, on a transit line of unremarkable length, gives approximately one thousand trillion, or approximately one quadrillion timetabling options to choose from. Like the normal case of transit vehicle scheduling, the special case of express-service BRT timetabling can be clearly seen to be 'suffering' from a combinatorial 'explosion.' As such, the vehicle scheduling problem and the express-service BRT timetabling problem have, in this regard, more in common with each other, than

120 *Key Facts, Transport for London Website*, Available: http://www.tfl.gov.uk/corporate/modesoftransport/londonunderground/1608.aspx# (Accessed 19 February 2009)

either has with the normal timetabling problem, which is not a combinatorial problem and can be solved via a simple formula.

As discussed in the last section, modern approaches to solving combinatorial problems involve searching through possible solution options so as to, in small cases, find the optimal solution, or in large intractable cases, find a good solution using a finite amount of computer power. As has been demonstrated above, express-service BRT timetabling is a combinatorial problem and so, as is the case with vehicle scheduling, the best solution to this problem is likely to be found by using computing power to search through the vast array of possible timetabling options, looking for the best, or at least a good solution. At minimum, the seeming commonality between normal vehicle scheduling and normal express-service BRT timetabling, represents a prima facie case for developing a software tool for express-service BRT timetabling based around combinatorial search methods.

There is no evidence of the existence of such a software tool, one specifically aimed at the case of express-service timetabling, in the literature. The TransMilenio S.A., operator of the TransMilenio BRT, does not use such a software tool, relying on semi-computerised methods that will be described later in this thesis.[121] With regards to the TransMilenio BRT, one article from 2008 states that “after more than 8 years it is still regarded as the gold standard for Bus Rapid Transit. Cities as diverse as New York, Delhi, Jakarta, Johannesburg, Beijing and Mexico, to name a few, have all drawn inspiration from Bogotá. But still there is no BRT

121 From notes taken at a meeting between the author and Alejandro Niño of the TransMilenio S.A. Planning Department on the 7th of June 2006.

system that matches its performance - at least not yet."[122] Not only does the "gold standard" BRT system not use a specific express-service bus timetabling software tool, but while attempting to acquire real-word BRT data for this thesis, a process that involved talking to numerous BRT system operators and BRT experts over a period of approximately one year, there was no hint of the existence of such a tool. Finally, a paper on a related topic to this thesis, *Design of Express Services for an Urban Bus Corridor with Capacity Constraints*, states:

> "The design of an express service involves defining its route and frequency as well as the size or type of buses to be used. Yet despite the importance of these factors for ensuring the system's efficiency, we found no published works in the literature that provide the necessary tools for creating a set of express services that would minimise the social costs of an urban bus corridor."[123]

All the evidence points to the conclusion that a specific express-service timetabling software tool does not exist.

Furthermore, it is not possible to just take transit scheduling software and apply it to the BRT timetabling problem. This chapter has gone to some lengths to show the conceptual similarity between BRT

122 *Why Is TransMilenio Still So Special?* (2008), The City Fix Website, Available: http://thecityfix.com/why-is-transmilenio-still-so-special/ (Accessed 17 January 2009)

123 Leiva, C., Muñoz, J. C. and Giesen, R. (2009), *Design of Express Services for an Urban Bus Corridor with Capacity Constraints*, Note: This paper by The Department of Transport Engineering and Logistics of the Pontificia Universidad Católica de Chile is, as of September 2009, in the second round referee stage for publication in Transportation Research B.

timetabling and normal transit scheduling, specifically that they are both combinatorial problems, but this similarity is limited to the conceptual level. Transit scheduling software is designed to take pre-determined service timetables and to generate efficient vehicle and driver schedules. By contrast, an express-service timetabling software tool would need to take, as its primary input, passenger load data and generate, as its primary output, bus run timetable data. With both the input and output data for an express-service timetabling software tool being different from that of existing transit scheduling software tools, there is no question of merely ‘plugging’ passenger data into existing transit scheduling software.

As well as the completely different input and output data-sets for transit scheduling software and any express-service timetabling software tool, the data processing required is completely different. Transit scheduling software packages process timetables into vehicle and driver schedules, and so their data processing involves concepts such as how to judge different options for joining up transit vehicle runs into work shifts, or staying within the maximum amount of time a driver can be on shift for. Neither of these concepts apply to the problem of generating efficient express service timetables. There is, however, a quite different set of concepts involved in processing passenger load data so as to generate efficient express-service timetables.

For example, how is passenger load to be represented? Passenger load is normally measured at the departure stop, with a given number of passengers arriving at a departure stop, within some specified time period, wanting to travel to a given destination stop. Buses, though,

provide passenger load servicing capacity along a transit line, not at a departure stop, as such. The manner in which these two different views of passenger load are reconciled has to be specified to construct a timetable, and this issue is a problem specific to timetabling. Another example of a timetabling specific problem, is the mix between all-stop and express bus services. For example, if it has been determined that twelve buses are required to service a given passenger load level, how many all-stop and how many express buses should there be? This problem of specifying the mix between all-stop and express services is also one that is specific to timetabling. Problems such as these need to be solved before any express-service timetabling software tool is in a position to begin assessing express-service stopping patterns, looking for an efficient choice. So before reaching the area of combinatorial commonality between BRT timetabling and transit scheduling, specific timetabling concepts need to be addressed.

Furthermore, even within this area of combinatorial commonality there are specific timetabling concepts that need to be addressed, in particular the question of the metric used to judge express stopping patterns. Transit scheduling software packages produce efficient driver and vehicle schedules, and so have metrics tailored to judge efficiency in these areas, not in the area of express service stopping patterns. Developing a metric to judge express service stopping patterns is not a simple matter. In a slight fore-shadowing of the work to be done in this thesis, section 5.7, “Making the Runs: Making Run Groups”, discusses the fact that the first, carefully considered, metric developed in this thesis to judge express service stopping patterns performed very poorly.

Section 5.7 goes on to explain that the ‘lessons learnt’ from the specific way in which the first metric performed poorly, allowed a second well-functioning metric to be developed.

So, even at the very centre of the conceptual similarity between BRT timetabling and transit scheduling, in the area of combinatorial searching, the differences between timetabling and scheduling yield complex timetabling specific problems that need to be solved. Outside of this area of combinatorial commonality, the timetabling and scheduling problems become radically different, with completely different input and output data-sets, and a very different set of problems that need to be solved. As a consequence, existing transit scheduling tools cannot be used to generate highly-efficient express-service timetables.

If a software tool to generate highly-efficient express-service timetables does not exist, and if the evidence suggests that such a tool might be able to produce better timetabling outcomes, then there is a case for developing at least a prototype of such a software tool to see how it performs. This thesis reports on research undertaken to develop such a prototype express-service timetabling software tool, and on how the resulting tool performs. Depending on that performance, not only might such a software tool be useful for existing express-based BRT systems such as the TransMilenio, but future express-based BRT systems, which may well be numerous given that the use of express services is becoming a recommended feature for BRT systems. For example, the California Partners for Advanced Transit and Highways’ report entitled *Framework for Bus Rapid Transit Development and Deployment Planning* says that “busway route structure should include a combination of basic

all-stop service that is complemented by express (or limited-stop), feeder, and connector service. The all-stop service can run all day, from about 6 a.m. to midnight, 7 days a week, and the express service should operate weekdays throughout the day or just during rush hours."[124]

It has been shown in this section that the express-service BRT timetabling problem is a combinatorial problem, and so in designing a software tool specifically for express-service BRT systems this needs to be taken into account. To mirror the problem's combinatorial nature, the software tool developed for this thesis will, in short, apply computing power to search through a myriad of express bus patterns looking for ones that very precisely match expected passenger load. As passenger load varies during the day, different express patterns will be required for different times of the day. As the objective of this thesis is to explore "ultra-efficient" BRT timetabling, the software tool reported on will change express patterns as quickly as the available expected load data changes, leading to a great many express patterns over the course of a one day timetable. So, in search of the most efficient outcome, not only will computing power be applied to generate highly-efficient express bus patterns, but those express patterns will be rapidly changed to match expected passenger load.

In conclusion, express-service BRT timetabling can be viewed as a special case of timetabling, in that, unlike normal timetabling, it is a combinatorial problem. Normal vehicle scheduling is also a combinatorial problem. Drawing from this commonality, the proposition has been

124 Miller, M. A., Yin, Y., Balvanyos, T. and Ceder, A. (2004), *Framework for Bus Rapid Transit Development and Deployment Planning* (Research Reports: Paper UCB-ITS-PRR-2004-47), California Partners for Advanced Transit and Highways, Berkeley, California, U.S.A., p. iv

advanced that express-service BRT timetabling might be most efficiently solved in the same basic manner as normal vehicle scheduling, namely via a software tool built around a recognition of the problem's inherently combinatorial nature. Given this one area of commonality, it has been discussed that there is little else similar between the problems of normal vehicle scheduling and BRT timetabling, and that a significant number of problems relating to BRT timetabling would need to be solved so as to be able to build an express-service BRT timetabling software tool.

The work of this thesis will be to address these timetabling specific problems, and to development of such a software tool. With such a software tool, the proposition that a combinatorial approach can timetable BRT more efficiently than a non-combinatorial approach, can be tested. The existing software tools used to solve the normal vehicle scheduling problem are, due to the combinatorial nature of that problem, sufficiently complex for one of their number to have earned the moniker "fearsome beast." With the express-service BRT timetabling problem also being combinatorial in nature, any software tool designed to solve this problem using a combinatorial approach, is also likely to be somewhat of a "fearsome beast." Given, though, that vehicle scheduling is a mature domain, whereas express-service BRT timetabling is only at an early research stage, the software tool developed here will only be a small, prototype "fearsome beast."

This section has addressed the second research question of this thesis, namely whether there is any theoretical evidence that current manual timetabling techniques are ill-suited to the task of producing highly efficient BRT timetables. For BRT systems which run express

services, side-by-side with an all-stop service, this question has been answered in the positive. In this section, an ordinary length transit line, running only one express service side-by-side with an all-stop service, has been shown to have over 30 million possible stopping patterns. Using current, essentially manual, timetabling methods only a tiny proportion of this vast number of possible stopping patterns would have their efficiency assessed.

2.7 Conclusion

This chapter has introduced both BRT systems and the transit timetabling process, so as to provide a solid basis for understanding the research presented in the remainder of the thesis. The key characteristics of BRT systems have been presented and discussed, and a definition of BRT has been built from these characteristics, with a 'classical' BRT system being defined as a bus-based transit mode, capable of servicing high passenger loads, that operates on bus-only roadways, and that has enhanced station environments, level boarding and pre-boarding ticketing. Unusual characteristics of the TransMilenio BRT have also been presented and discussed, so as to provide a full understanding of the characteristics of the test site for which an ultra-efficient timetable is to be generated.

The transit timetabling process has also been introduced, with the important aspects of this process being discussed in the broader context of the transit planning process, of which the transit timetabling process is a part. The combinatorial nature of normal vehicle scheduling has been discussed, and the fact that, due to its combinatorial nature, normal

vehicle scheduling is most efficiently solved using software tools designed specifically for this purpose. It has been argued that express-service BRT timetabling, unlike normal timetabling, is also a combinatorial problem, and consequently the thesis has been advanced that express-service BRT timetabling might also be best solved using a software tool designed specifically for that purpose. Beyond this one area of combinatorial commonality, the significant differences between normal scheduling and express-service BRT timetabling have been highlighted. Consequently, the building of a software tool to timetable BRT systems in a combinatorial manner requires that a substantial number of timetabling specific problems be solved. This thesis will aim to solve these timetabling specific problems, build such a software tool, and thereby be in a position to test the proposition that BRT can be more efficiently timetabled in a combinatorial manner.

Chapter 3

The TransMilenio's Américas Line

3.1 Introduction

This thesis reports on the development of an ultra-efficient timetable for one line of the TransMilenio system, not the system as a whole, and so it is necessary to 'separate' one of the TransMilenio lines from the rest of the system. A peculiarity of the TransMilenio system is that, unlike most metro systems, there is little direct correspondence between the TransMilenio's transit services and its physical lines. Of the 16 weekday bus services running on the TransMilenio, as of May 2005, only six stayed 'bound' to one physical line. As the TransMilenio bus services do not stay 'bound' to one physical line, neither do passenger trips; passengers can and do move from one physical line to another while staying on the same service.

Of the three major physical lines of the TransMilenio system, as of May 2005, the Américas Line is the least effected by this issue, with two of its four services staying 'bound' to it. The Américas Line is also only marginally longer than the shortest of the major lines, having 18 stops compared to the shortest major line's 15 stops. With this thesis examining a combinatorial timetabling problem, where the number of timetabling options goes up exponentially in relation to the number of stops, there is a good reason to favour one of the two comparatively short lines over the long major line, which has 42 stops. For these two reasons, that a comparatively high number of the Américas Line's services stay bound to its line, and that it is a relatively short line, the Américas Line

was selected as the line to be used for the development of an ultra-efficient timetable.

This chapter presents how the bus service and passenger load data of the Américas Line is isolated from the TransMilenio system as a whole. The TransMilenio bus service and passenger load data used throughout this thesis was very kindly provided by TransMilenio S.A., the management authority for the TransMilenio BRT system.[125] TransMilenio S.A. was in a position to provide the bus service and passenger load data as two of their key areas of responsibility are the collection and collation of passenger load data and the construction of bus service timetables to meet expected passenger load. In outlining the process of isolating the Américas Line bus service and passenger load data, this chapter begins by describing the physical characteristics of the Américas line. The passenger load data is then presented, first in its non-isolated form and then in its isolated form, along with the method use to isolate it. Finally the bus run data is presented, again first in its non-isolated form and then isolated along with the isolation method. With regards to the passenger load and bus service data isolation, the methods used to isolate the data is presented down to the level of fine technical detail. Once the data for the Américas Line is isolated an initial assessment of this data is conducted, to determine whether there appears to be an opportunity to achieve efficiency gains via ultra-efficient timetabling.

125 *TransMilenio Busway-Based Mass Transit, Bogotá, Colombia*, The World Bank Website, Available: http://siteresources.worldbank.org/INTURBANTRANSPORT/Resources/Factsheet-TransMilenio.pdf (Accessed 17 January 2009), p. 11

3.2 The Physical Characteristics

Although much of this thesis focuses on the 'logical' aspects of timetabling BRT systems, using the TransMilenio BRT system as a test case, the timetable development in this thesis does not exist in isolation, but is drawn from a physical BRT system. Below is a schematic diagram of the system.

Plate 3.1: Schematic Diagram of the TransMilenio BRT System

In Plate 3.1 above the Américas Line is shown in green. The Américas line is approximately 13.3 kilometres long and has 18 stops, including the stops at either end of the line. The location of the Américas Line stops is shown below, with stop distances being measured from the Avenue Jiménez stop, which is located adjacent to Avenue Caracas, the busiest of the TransMilenio corridors.

Stop Name	Distance (metres)
Avenue Jiménez	0
De La Sabana	809
San Facon - Cr. 22	1,502
Ricaurte	1,992
CDS - Cr. 32	2,583
Zona Industrial	3,279
Cr. 43	3,750
Puente Aranda	4,176
Américas - Cr. 53A	5,269
Pradera	6,538
Marsella	7,612
Mundo Aventura	8,472
Mandalay	9,171
Banderas	9,951
Tv. 86	10,823
Biblioteca Tintal	11,576
Patio Bonito	12,292
Portal de las Américas	13,296

Table 3.1: Location of the Américas Line Stops[126]

The average distance between stations on the Américas Line is 782 metres. This station spacing is tighter than is common for normal subway services but wider than is common for normal bus services. Various paper have been written about bus stop spacing, particularly focusing on the question of optimal stop spacing. For example, a paper entitled *Assessment of Optimal Bus Stop Spacing Model Using High Resolution Archived Stop-Level Data*, calculates the optimum bus stop spacing for an example line in Portland, Oregon at 372 metres, approximately half the spacing distance of the Américas Line.[127] It is, though, difficult to know how applicable such a figure might be, as the Portland, Oregon line is a

126 This stop spacing data is from a document named “data_bogota_spanish (2)-1” received by the author from the TransMilenio S.A.

127 Li, H., Bertini, R. L. (2009), *Assessment of Optimal Bus Stop Spacing Model Using High Resolution Archived Stop-Level Data*, In: Transportation Research Board, *Transportation Research Board 88th Annual Meeting Compendium of Papers DVD*, Washington, D.C., U.S.A., p. 13

normal bus line, whereas the Américas Line, is part of a trunk/feeder system, with the Américas Line being one of the trunk lines.

Another example paper, *Importance of Objectives in Urban Transit-Network Design*, addresses the broader question of "what are the optimal values for stop spacing and for line spacing in urban transit networks given traveller preferences and supply-budget constraints?"[128] This paper gives 800 metres as the optimal stop spacing for their "large city" example, when assessed on the basis of minimising travel time with a fixed operational budget,[129] a figure very close to 782 metre stop spacing of the Américas Line.

The maximum speed along the Américas route corridor is 60 km/h, though due to a substantial number of traffic lights the actual service speed is substantially lower, even in the case of express services that stop at only few stations. For example, express service 120, which stops at only 6 of the Américas Line's 18 stops, has a service speed of 29.5 km/h, a speed approximately half that of the maximum speed.[130] There are a total of 24 traffic lights along the Américas route, 16 of which are combined car/pedestrian lights and 8 of which are pedestrian only lights.[131] With its 13.3 kilometre length, this gives an average traffic light spacing of one approximately every half a kilometre. There is no traffic light pre-emption used by the TransMilenio system, so the services are fully impacted by these traffic lights.[132] Combined with the relatively tight

128 van Nes, R. and Bovy P. H. L. (2000), *Importance of Objectives in Urban Transit-Network Design*, Transportation Research Record, Volume 1735, p. 25

129 op. sit., p. 32

130 From bus timetable data received by the author from the TransMilenio S.A.

131 Count taken by the author during June 2006 site visit.

132 From notes taken at a meeting between the author and Alejandro Niño of the TransMilenio S.A. Planning Department on the 7th of June 2006.

station spacing, the high frequency of non-pre-empted traffic lights explains why even the express buses, which stop at only a few stops, are only able to achieve a service speed approximately half that of the line speed.

As is common with various different types of transit systems, the passenger load at stops along the TransMilenio's Américas Line varies considerably. With rail-based transit systems, this load variation is usually not reflected in the length of station platforms: although the quantity of other facilities, such as ticket machines, shops and toilets, often varies significantly in line with expected load, with rail-based transit, as a consequence of the limitations on one train passing another, the length of stations does not commonly vary. Due to the fact that TransMilenio buses stop off the transit line, stops can be effectively bypassed; the length of stops therefore varies in rough accordance with expected peak demand. The length of stops is shown below in Table 3.2.

Stop Name	Stop Length (metres)
Avenida Jiménez	144.4
De La Sabana	110.8
San Facon - Cr. 22	144.4
Ricaurte	202.4
CDS - Cr. 32	110.8
Zona Industrial	110.8
Cr. 43	110.8
Puente Aranda	103.6
Américas - Cr. 53A	130.0
Pradera	103.6
Marsella	195.2
Mundo Aventura	130.0
Mandalay	130.0
Banderas	300.0
Tv. 86	130.0
Biblioteca Tintal	308.4
Patio Bonito	308.4
Portal de las Américas	Not Applicable

Table 3.2: Length of the Américas Line Stops[133]

There is a degree of similarity in the above stop length figures, reflecting the standard components used to build the stops, and a degree of dissimilarity, reflecting the impact of specific local conditions on the stop design. The final stop on the line, Portal de las Américas, does not have a stop length, as such, as it is a terminus, rather than as a stop. The ends of the TransMilenio corridors have terminus stops, where feeder buses drop off passengers from points beyond the TransMilenio coverage area so that they can connect with TransMilenio BRT services. As at the end of Phase I of the Transmilenio's implementation, in 2002, there were "470 articulated buses and 235 feeder buses."[134]

133 This stop length data is from a document named "data_bogota_spanish (2)-1" received by the author from the TransMilenio S.A.

134 *TransMilenio's Contributions to the Development of Bus Rapid Transit Systems: by Darío Hidalgo Guerrero, Ph.D.*, Bogotá Lab Website, Available: http://www.bogotalab.com/articles/transmilenio.htm (Accessed 26 May 2008)

3.3 The ‘Raw’ Bus Runs Data

One possibility for this thesis would have been to construct bus run data with which to do the ultra-efficient BRT timetable development exercise. Such ‘synthetic’ data could have been derived from similarly notional transit line servicing a posited passenger load. Although for some types of theoretical research such an approach would have been perfectly acceptable, for the type of applied research being conducted here, real-world data, from a real-world transit system, was considered essential.

With the decision to use real-world data came the not-unexpected requirement to do some data conversion, so as to get the real-world data into a form appropriate for conducting a timetable development exercise. This section will present an overview of the ‘raw’ bus run data, leaving the task of explaining the how the bus run data has been processed to section 3.4.

A single spreadsheet file was received from TransMilenio, which contained every scheduled bus movement for the timetable that was being run during the Monday to Friday period. As is common with mass transit systems, the timetables run on Saturday, Sunday and public holidays differ from those that are run during the Monday to Friday period. This timetabling exercise has only concerned itself with the Monday to Friday period, specifically the five-day period of Monday the 23rd to Friday the 27th of May 2005. A sample of the bus movement data is given below.

TypeDay	Movement	ServBus	Event	Macro	Line	Car	Sub-line	Section	Point	Type	Trip
00101508	04:40:00	508001	3	1	0	1	0	0	1	5	0
00101508	04:47:00	508006	3	1	0	6	0	0	1	5	0
00101508	04:54:00	508009	3	1	0	9	0	0	1	5	0
00101508	04:54:30	508002	3	1	0	2	0	0	1	5	0
00101508	04:55:00	508003	3	1	0	3	0	0	6	5	0

Table 3.3: Sample of Bus Movement Data

The data in Table 3.3 above is used to run a transit system, not to conduct a timetable development exercise, and so, unsurprisingly, much of this data is, for the purposes of timetable development, either duplicated or redundant. The important columns above are the Movement, Event, Line, Car and Point.

The Line column in Table 3.3 contains the bus service number, and has been used to separate this table into its respective bus services. As at May 2005 there were 16 services operating on the TransMilenio system on weekdays, four all-stop services and 12 express services. Of these sixteen services, four travel along the Américas Line, one all-stop service and three express services. Of these four Américas Line services, the all-stop service and one of the express services stays ‘bound’ to the Américas Line, whereas the remaining two express services run beyond the bounds of the Américas line.

The remaining four columns of interest in Table 3.3 are Movement, Event, Car and Point. Movement is simply the time at which a given event occurs. Event is the event that has occurred at that time, with the number codes referring to events such as “leave depot” and “arrive at stop.” Car is the physical bus that is making the event and Point is the stop at which the event is occurring, again given as a number code. So, to translate the key data from above columns in to simple terms: a given

event occurs to a given bus at a given place and time. Although somewhat buried, for the purposes of this timetable development exercise, in superfluous operational data, the event based data provided by TransMilenio S.A. is very comprehensive.

3.4 Isolating and Constructing the Bus Runs

The timetable development exercise being conducted here is only concerned with the Américas Line, and so the two express services that travel beyond the Américas Line need to be shortened. Rather than 'allowing' the buses to continue beyond the Américas line it has been necessary, in effect, to 'turn the buses around' at Avenida Jiménez, the last stop on the Américas line. To truncate the two express services which travel beyond the Américas Line to the Avenida Jiménez stop, their arrival time needs to be known, which means their speed, or at least an estimate of their speed needs to be determined.

One option for determining the express services' speed would be to use the average speed for express buses on the TransMilenio as a whole, reported at 32 km/h.[135] For express service number 100, which travels the entire 13,296 metre length of the Américas Line, this would give a travel time of 24 minutes and 56 seconds. For express service number 80, which only travels from the Banderas stop, 9,951 from Avenida Jiménez, this would give a travel time of 18 minutes and 39 seconds. The problem with this approach is that the 32 km/h express speed figure is for the

135 *TransMilenio's Contributions to the Development of Bus Rapid Transit Systems: by Darío Hidalgo Guerrero, Ph.D.*, Bogotá Lab Website, Available: http://www.bogotalab.com/articles/transmilenio.htm (Accessed 26 May 2008)

TransMilenio system as a whole and does not necessarily reflect the specific conditions on the Américas Line.

Another approach to calculating the speed would be to use the single express service on the Américas Line, service number 120, that does not travel beyond the Américas Line as a gauge to the speed of express services on the Américas Line. Express service 120 travels the entire 13,296 metre length of the Américas Line and stops at 6 stops, which gives it an average spacing between stops of 2,659 metres. One of the two services which need to be truncated, service 100, also travels the entire length of the line and stops at 6 stops, and therefore also has an average spacing between stops of 2,659 metres. The other service which need to be truncated, service 80, travels 9,951 metres of the Américas Line and stops at 5 stops, which gives it an average spacing between stops of 2,488 metres, only slightly less than the average spacing of the other two services. Due to similar stop spacing, and the fact that express service 120 provides a speed gauge specific to the Américas Line, this express has been chosen to determine a line speed with which to truncate the other two express services.

From the data received, the average time taken for express service 120 to travel the 13,296 length of the Américas Line is 27 minutes. Applied to express service 100, this time is two minutes and four seconds longer one-way, or four minutes and eight seconds longer two-way, than the time given by the average express speed discussed previously. With the maximum bus frequency, from the data, of one bus every three minutes, this difference would generate an extra bus drain of between one and two buses.

To do the same calculation as above for express service 80, the 27 minutes taken by express service 120 to travel the 13,296 length of the Américas Line, need to be scaled down to the distance travelled by express service 80, namely 9,951 metres. This scaling down yields a time of 20 minutes and 12 seconds. Applied to express service 80, this time is (18 minutes and 39 seconds) one minute and 33 seconds longer one-way, or three minutes and six seconds longer two-way, than the time given by the average express speed discussed previously. With the maximum bus frequency, from the data, of one bus every two minutes, this difference would generate an extra bus drain of between one and two buses.

In choosing between the two time methods above, using the average system speed and using the Américas Line express 120 as a gauge, the second method is strongly favoured as it represents data specific to the Américas Line. The average system speed though can act as a useful double check on the approach of using express 120 as a gauge. For both service 80 and 100, the overall system speed approach gives somewhat faster travel times than the specific Américas Line express 120 approach. For the overall system speed approach to confirm the slower travel times of the specific Américas Line express 120 approach, it would need to be the case that their was clear evidence that the TransMilenio system as a whole ran at higher average speeds than the Américas Line.

There is such evidence, in that although the notional TransMilenio operating speed is consistent across the network, the distribution of traffic lights is not, due to the uneven distribution of crossing roadways. Parts of the TransMilenio system, notably the northern stretches are, by comparison to the Américas Line, relatively free of crossing roadways

and therefore traffic lights. The uneven distribution of crossing roadways is observable on Plate 3.2 below, a TransMilenio system map from a somewhat later date than the May 2005 date of this research.

Plate 3.2: Map of the TransMilenio BRT System[136]

With a lower distribution of traffic lights on parts of the TransMilenio system other than the Américas Line, it is to be expected that the average TransMilenio system speed would therefore be somewhat higher than the speed of the Américas Line. The two different methods used to calculate the express line speed here, although they yield slightly different nominal results, do therefore in fact confirm one another.

Having calculated a reliable travel time for the two express services that need to be shortened, the bus service along the Américas Line can be trimmed to the Avenida Jiménez stop. For the outbound buses, this

136 *Rutas Transmilenio Mapa Bogotá*, Ciudad Movil Website, Available: http://www.ciudadmovil.com.co/q/mod/mapa/bogota.php (Accessed 17 February 2005)

was done by adding the time services took to reach the Avenida Jiménez stop after departing the Portal de las Américas or Banderas stops, and truncating the service by adjusting the stop and arrival time records. For example, a bus departing the Portal de las Américas stop at 10:00 had its data record modified such that it arrived at the Avenida Jiménez stop 27 minutes later, the time taken to travel the line. The outbound bus services were similarly truncated, by calculating how long before the buses reached the Portal de las Américas stop they were at the Avenida Jiménez stop, and similarly modifying the data records.

By this method the two services which extended beyond the Américas Line have been truncated to the Américas Line, in a manner that delivers sound approximate bus run times. This method does, though, as is often the case, solve one problem while creating another, that relates to the turn around time of the services. In their pre-truncated form, express services 80 and 100 were designed to have efficient turn-around times at their pre-truncated end stops. By cutting these services at Avenida Jiménez, what is in effect for these pre-truncated services an arbitrary mid-line stop, this efficient allocation of turn-around time has been lost. It might be said that the ‘neatly-tied’ ends of the original timetable have, as a consequence of being cut at an ‘arbitrary’ point, become ‘frayed’ ends. Unless this issue is addressed, buses will wait at their end-stop longer than is actually a fair representation of the TransMilenio system and therefore a higher than true bus drain level will be reported.

One way of dealing with this issue would be to modify the start times of the bus runs in such a manner that the ‘frayed’ ends at Avenida Jiménez inner-most stop match up. The objective of such start-time

modifications would be, in effect, to try and mimic the timetable TransMilenio S.A. would have generated if they had themselves designed the timetables to stop express service 80 and 100 at Avenida Jiménez. The primary problem with this approach is computational complexity, particularly given that express service 80 is linked to Portal de las Américas, the outer-most stop, using limited deadhead runs. An alternative approach to addressing this ‘frayed’ ends issue is to treat the buses that end up waiting at Avenida Jiménez, the inner-most stop, as computational artefacts, to be compensated for rather than to be eliminated. With regards to this case, compensating for these artefacts means treating the Avenida Jiménez as part of the main depot pool of buses, thus eliminating their effect on the bus drain levels. Primarily for reasons of simplicity, this later approach has been used.

3.5 The ‘Refined’ Bus Runs Data

Although the calculations required to generate ‘refined’ bus run data were somewhat complicated, the resulting data is itself rather straightforward. Of interest is the total number of buses required to provide the services running along the Américas Line, and the bus drain level across the day. This data is presented below.

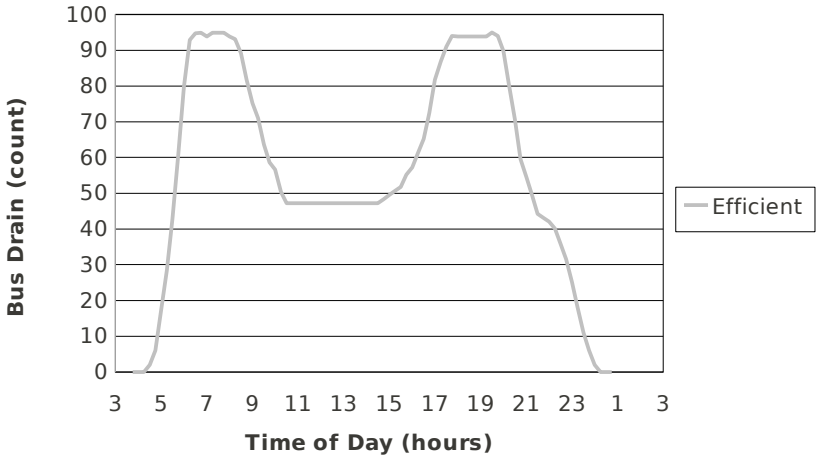


Figure 3.1: Bus Drain by Time of Day

Method	Efficient
Maximum Bus Drain (count)	95

Table 3.4: Maximum 24-Hour Bus Drain

Of note in Figure 3.1 are the clear flat areas, visible most distinctly during the middle-part of the day, and the equally clear angular transitions between different the flat areas. Passenger load data does not tend to exhibit such angular characteristics, and so their presence here can be seen as an aspect of the timetabling method employed, an issue that will be examined further is section 3.9, “Comparing the Bus Run and Passenger Load Data.”

At 95 total buses for a 13 kilometre route it is clear that the Américas Line is a high-capacity route. With a turn around time on the Américas Line of approximately one hour, the peak overall bus dispatch rate, during the morning and afternoon peaks, is approximately one bus every 40 seconds. Given that these buses are 18 metres long and have a rated capacity of 160 passenger, this dispatch rate indicates a very high

passenger load level; a matter that will be covered in detail in the next few next sections on passenger load data.

3.6 The ‘Raw’ Passenger Load Data

As with the bus run data, ‘raw’ passenger load data was received from TransMilenio S.A., which has required processing before being usable for the purpose of developing an ultra-efficient BRT timetable. Two quite separate data-sets, relating to passenger load, were received from TransMilenio S.A.

The first data-set consisted of five structurally identical spreadsheets, covering the same five-day period of Monday the 23rd to Friday the 27th of May 2005 as the run data. These spreadsheets contain the number of passengers that enter and leave each stop on the TransMilenio, for the entire day, with the data being given in 15-minute increments. A sample of this data is given below.

Station Code	Entrance Code	Initial Date/Time	Final Date/Time	Entry/Exit	Magnitude
12000	11	23/05/2005 5:15	23/05/2005 5:29	E	1
12000	11	23/05/2005 5:30	23/05/2005 5:44	E	4
12000	11	23/05/2005 5:45	23/05/2005 5:59	E	8
12000	11	23/05/2005 6:00	23/05/2005 6:14	E	13
12000	11	23/05/2005 6:15	23/05/2005 6:29	E	17

Table 3.5: Sample of Passenger Movement Data

Apart from the translation of the column headings from Spanish to English, the sample of data presented above is exactly as received. The station code column is self explanatory; the station codes were translatable into station names via information available in the second passenger data-set received from TransMilenio S.A. As is common on various transit systems, the larger of the TransMilenio stops have more

than one entrance, with the codes for these entrances being recorded in the second column. No use was made of the entrance code data. The Initial Date/Time and Final Date/Time columns are likewise self-explanatory. For the Exit/Entry column the letter “E” denotes entry, as entry is “entrada” in Spanish, and the letter “S” denotes exit, as exit is “salida” in Spanish. The magnitude column is simply the number of people during the specified time period.

The TransMilenio is a substantial system with, as at May 2005, 79 stops, as observable in Plate 3.1. Add to the fact that there are 79 stops, the fact that the data is presented in 15 minute increments, and the first passenger data-set of substantial in size. Each of the five spreadsheet have approximately 42 thousand rows of data, so for the complete Monday to Friday period approximately two hundred thousand passenger-movement data points were received. Fortunately, such a data load does not present a significant challenge to the computational capabilities of modern computers.

The second passenger data-set received from TransMilenio S.A. were three origin-to-destination spreadsheets, for the time periods 06:45 to 07:45, 12:00 to 13:00 and 17:15 to 18:15. Although this data is not comprehensive, in that it does not cover all time periods, it is fairly representative in that it covers the morning peak, the midday lull and the afternoon peak. A sample of this data is given below.

06:45 - 07:45		Puente Aranda	Carrera 43	CDS - Carrera 32	Ricaurte
	E/S	12000	12001	12002	12003
Puente Aranda	12000	0	0	2	3
Carrera 43	12001	0	0	0	1
CDS - Carrera 32	12002	1	4	0	0
Ricaurte	12003	2	2	1	0

Table 3.6: Sample of Passenger Origin-to-Destination Data

The above table is fairly self explanatory. The stations are named on the top row and leftmost column, and on the second row down and second column from the left the station codes are given. The rows represent passengers entering the stops and the columns passengers exiting the stops, a circumstance which, given the potentially ambiguous “E/S” label, was double checked by checking the data against the mass inbound and outbound flows. The numbers above are simply the number of passengers that are entering and exiting a given directed stop pair.

The ‘raw’ passenger data, received from TransMilenio S.A. and presented in the section, is of a very fine granularity, compared to that which may have been available. The ideal passenger data-set would have been time-stamped individual passenger origin-to-destination record, from which any other needed data-set could have been constructed. Although such a data-set was not available, the passenger data which the TransMilenio S.A. very kindly provided, is more than sufficient for which to isolate and construct the data needed to develop an ultra-efficient timetable.

3.7 Isolating and Constructing the Passenger Load

To develop an ultra-efficient BRT timetable it is necessary to know when passengers arrive at a stop and which stop they are travelling to, in

other words passenger origin-to-destination data. To develop an ultra-efficient timetable for only one line of a larger system, it is necessary to have data about just the one line not the whole system. The data received from TransMilenio S.A. is close to but not quite passenger origin-to-destination data and it is not separated into distinct lines. This section shows the process that was used to convert the data received from TransMilenio S.A. into passenger origin-to-destination data and how the Américas line was separated from the rest of the TransMilenio BRT system.

Of the two data-sets received from TransMilenio S.A. only one, the stop entrance and exit data, covers the entirety of the day, and the required Monday to Friday period. This data-set has therefore been used as the primary source from which to construct the passenger load matrix. The entrance and exit data for the stops not only covers the required time periods it also covers all of the stops on the TransMilenio system. Before the issue of data construction can be addressed though the Américas Line has to be separated from the rest of the system. The only passengers that are of interest are those that travel down the Américas Line corridor, in other words between one of the 18 Américas Line stops. The ‘links’ between the stops on the Américas Line, and near surrounds, are shown in Figure 3.2 below.

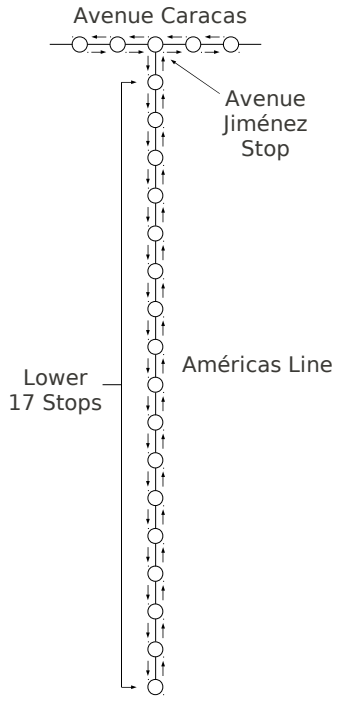


Figure 3.2: Inter-stop Links Between Stops on the Américas Line

The 18 Américas Line stops fall into two groups, the lower 17 stops, and the uppermost stop, Avenue Jiménez. For the lower 17 stops, any passenger who enters or exits one of these stops must be about to travel along, or have just travelled along, the Américas Line corridor. As can be seen in Figure 3.2 there are no ways of arriving or leaving the lower 17 stops without travelled along the Américas Line corridor. Passengers entering or exiting one of the lower 17 stops, may or may not travel beyond, or have just travelled beyond, the Américas Line corridor, but there is no doubt that they travel on the Américas Line corridor.

The same cannot be said for Avenue Jiménez, the uppermost stop. A passenger entering Avenue Jiménez, may be travelling along the Américas Line corridor, or may not be. Similarly, a passenger exiting Avenue Jiménez, may have just travelled along the Américas Line corridor, or may not have. As can be seen in Figure 3.2 there are three routes in and out of Avenue Jiménez, and only one of them is along the

Américas Line corridor. In this sense, Avenue Jiménez is the same as all of the other stops on other corridors of the TransMilenio system, in that a passenger entering or exiting these stations may or may not have travelled on the Américas Line corridor.

As part of isolating the passenger data, the stops have been grouped as indicated above, with each stop being either one of the lower 17 stops on the Américas Line or one of the rest-of-the-stops, which for simplicities sake will be referred to as the upper-part stops. When a passenger enters one of the upper-part stops, their trip may have them exiting at one of the lower 17 stops on the Américas Line, or it may have them exiting at one of the upper-part stops. With regards to isolating the Américas Line corridor, only those trips that exit one of the lower 17 stops are of interest. Similarly, when a passenger exits one the upper-part stops, their trip may have started with them entering one of the lower 17 stops, or it may have started with them entering one of the upper-part stops; only those trips that start at one of the lower 17 stops is of interest here.

The passenger data though is not so neatly separated, only a passenger's entry or exit is recorded, not their origin/destination. To separate out the Américas Line corridor passengers from the rest, some origin/destination assignation needs to be made. This assignation has been done using the second set of passenger data received from TransMilenio S.A., the three origin-to-destination spreadsheets. These spreadsheets provide origin-to-destination data for three different periods during the day, the morning peak, the midday lull and the afternoon peak. For the purposes of this data construction exercise these

three spreadsheets have been averaged to provide a single origin-to-destination table.

An origin-to-destination table allows probability questions about a passenger's trip to be posed and answered. Specifically, when a passenger enters one of the upper-part stops, it is possible to calculate the probability that they will exit at one of the lower 17 stops. This can be done by dividing the sum of the trips which enter at a given upper-part stop and exit one of the lower 17 stops, by the sum of all trips which enter that given upper-part stop. A similar calculation can be made to calculate the probability that if a passenger enters one of the lower 17 stops they will exit one of the upper-part stops.

If passenger travel off the Américas Line corridor were of interest, the specific stops passengers boarded or alighted from would be of interest, and their individual probability levels would need to be calculated. It is the case though that travel off the Américas Line corridor is not off interest and so all trips, no matter which upper-part stop they start or end at, will be, in effect, truncated to the Avenue Jiménez stop. As a consequence, with regards to this exercise all of the entry and exit data for the upper-part stops can be added together and treated as data for the Avenue Jiménez stop.

Rather than a very large number of probabilities only two are actually needed; if a passenger enters any upper-part stop what is the probability that they will exit a lower 17 stop and conversely, if a passenger exits any upper-part stop what is the probability that they entered a lower 17 stop. From the data provided, the number of passengers that entered an upper-part stop and exited a lower 17 stop

was 17,137 out of a total of 219,242, giving a probability of approximately 7.8%. Conversely, the number of passengers that exited an upper-part stop and entered a lower 17 stop was 22,920 out of a total of 225,025, giving a probability of approximately 10.2%.

By truncating all of the upper-part stops to the Avenue Jiménez stop, and then, through the use of origin-to-destination probability data, removing all non Américas Line data, isolated stop entry and exit data has been constructed. For the purposes of timetable development though it is not stop entry and exit data that is required but inter-stop load levels.

To convert stop entry and exit data into inter-stop load levels the entry and exit data has to, in effect, be paired; a given rate of entry at one stop has to be matched to a given rate of exit at another. Entry and exit data though are happening in slightly different time-frames, with the time of a bus trip separating the exit data from the entry data. Although this is not a matter of the greatest significance, as the magnitude of the data is going to be adjusted anyway, some rough attempt has been made to bring the two data-sets into the same time-frame. The time taken to travel down the full length of the Américas Line is approximately 30 minutes; half this time, namely 15 minutes, was subtracted from all exit data to roughly compensate for the trip time. In addition, another 15-minutes was subtracted to exit data from the upper-stops, as the trip time from the Avenue Jiménez stop to other parts on the TransMilenio system is of the order of 30 minutes. As a mirror of this change, 15-minutes was added to entry data from the upper-stops.

Having adjusted the entry and exit data into the same time-frame the entry and exit data can be converted into load data. For each 15-minute time period there are 18 entry load levels and 18 exit load levels. To convert one entry level, say for the Portal de las Américas stop, into 18 load levels the probability of a passenger who has entered Portal de las Américas exiting a given other stop is taken to be the rate at which passengers are exiting that stop compared to the other stops. So, for example, if one passenger was exiting the Avenue Jiménez stop out of 100 passengers exiting in total, then it would be assumed that only 1% of the passengers entering a stop were exiting at Avenue Jiménez. The case of passengers exiting from the entry station under consideration is excluded, as it is assumed that passengers entering a stop do so to travel to a different stop.

The above calculations isolate the passenger data and construct inter-stop load levels from-stop entry and exit data, as well as from limited inter-stop load level data. If precise passenger magnitude levels were required for this ultra-efficient timetable development exercise, then the degree of data manipulation above might be beyond what would be prudent. It is the case though, that the purpose of the exercise here is merely to construct a pattern of load across the day which fairly represents the situation on the Américas Line. To complete the passenger data construction, the passenger load data has been scaled up so that the boardings equal the projected boardings level of 186,000 per day.[137,138]

137 Cain, A. (Principal Investigator) (2006), *Applicability of Bogotá's TransMilenio BRT System to the United States - Final Report May 2006*, Federal Transport Administration, Department of Transportation, United States of America, Washington, D.C., U.S.A., p. 20

138 *Bogotá, What's Next?: by Dario Hidalgo* (2006), EMBARQ: The World Resources Institute Centre for Sustainable Transport Website, Available: http://www.embarq.wri.org/ documentupload/WRI %20WB%20Jan%202006%20Dario%20Hidalgo.pdf (Accessed 17 February 2009), p. 7

This section, in conjunction with section 3.4, “Isolating and Constructing the Bus Runs”, has addressed the third research question of this thesis, namely whether data for one line of a BRT system can be easily separated from the data from the entire system. This question has been answered in the negative. Although one line of a BRT system can be separated from the entire system, this separation can only be achieved with difficulty. Two lengthy sections of this chapter, this section and section 3.4, were required to detail the process of separating the Américas Line of the TransMilenio BRT from the system as a whole.

3.8 The ‘Refined’ Passenger Load Data

Having isolated the passenger load data it is now possible to show the basic passenger load characteristics of the Américas Line. Figure 3.3 below shows the passenger load separated into its inbound and outbound components.

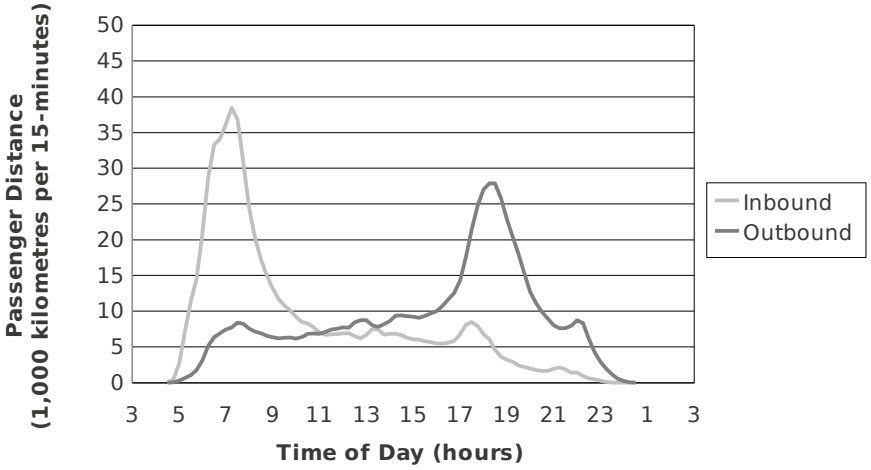


Figure 3.3: Américas Line Inbound and Outbound Passenger Distance by Time of Day

Direction	Inbound	Outbound
Total Passenger Distance (1,000 kilometres)	688	722

Table 3.7: Total Américas Line 24-Hour Inbound and Outbound Passenger Distance

The load on the Américas Line is in a classic double-peak formation, with the load on the inbound line peaking in the morning and load on the outbound line peaking in the afternoon. The morning peak is substantially higher than the afternoon peak as well as being somewhat narrower. Total load is very similar between the inbound and outbound line, with the outbound line having only a slightly higher total load.

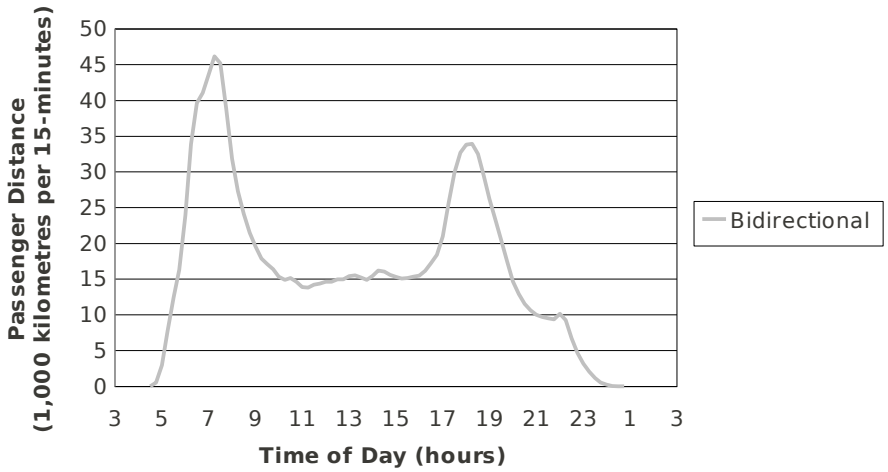


Figure 3.4: Américas Line Bidirectional Passenger Distance by Time of Day

Direction	Bidirectional
Total Passenger Distance (1,000 kilometres)	1,410

Table 3.8: Total Américas Line 24-Hour Bidirectional Passenger Distance

The above combined inbound-and-outbound load level is being generated by 186,000 boardings, the projected weekday rider-ship. The actual daily load levels for the Américas Line are shown in Table 3.9 below.

Day	Load Level (boardings)
Monday	180,844
Tuesday	181,363
Wednesday	184,692
Thursday	180,210
Friday	182,831

Table 3.9: Daily Passenger Load Level on the Américas Line

As can be seen in Table 3.9 above, the actual passenger load levels on the Américas Line, during the week the data comes from, were very close to capacity, with the Wednesday passenger load level being approximately 99% of capacity. This data reinforces the view of Alejandro Niño, of the TransMilenio S.A. Planning Department, that at around this time the TransMilenio system was running very near to capacity.[139] The timetabling in this thesis was conducted at the Américas Line's rated capacity of 186,000 boardings per day, with the data scaled up as necessary.

3.9 Comparing the Bus Run and Passenger Load Data

The purpose of bus runs is to service passenger load. In a system such as the TransMilenio, where the bus fleet is homogeneous, it could be expected, in the simple case, that with a higher level of passenger load there would be a greater number of buses servicing that load. In terms of servicing passenger load though, buses are only as good as their occupancy level; an empty bus services no load. Occupancy can rise and fall under the influence of both unavoidable structural issues as well as due to timetable planning choices.

139 From notes taken at a meeting between the author and Alejandro Niño of the TransMilenio S.A. Planning Department on the 7th of June 2006

The key unavoidable structural issue, with regards to occupancy, is the nature of the passenger load pattern. In particular, a significant imbalance between inbound and outbound loads can cause a drop in average occupancy levels, as buses are back-run empty or near empty. Occupancy levels may also be deliberately reduced to increase passenger comfort; except during the absolute peak time this can be done, to a greater or lesser degree, without increasing the size of the bus fleet. Finally, and most topically for this thesis, occupancy can varied as consequence of efforts to generate a timetable that is substantially less complex than the passenger load pattern it serves; something has to give and this something is occupancy.

In the preceding sections of this chapter the factors that make up occupancy, bus runs and passenger load, have been presented for the Américas Line. On Figure 3.5 below, these two levels are shown together.

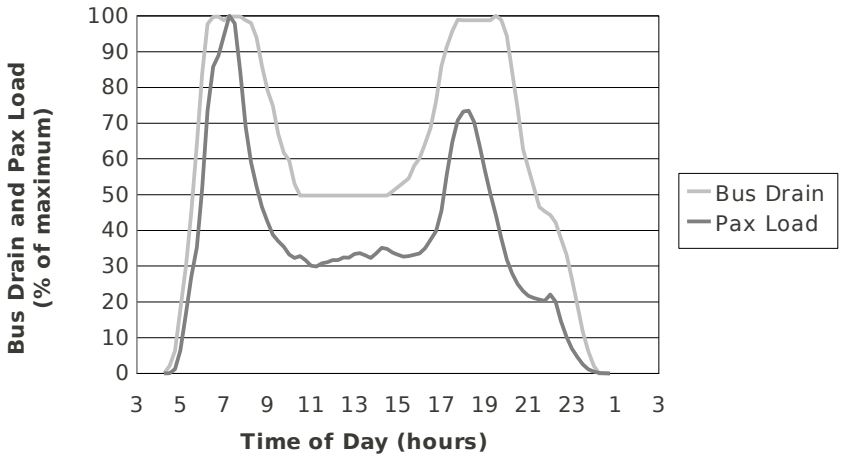


Figure 3.5: Bus Drain and Pax Load Compared by Time of Day

In Figure 3.5 above both the bus drain and passenger load levels have been scaled to be a percentage of their respective peak levels. With the purpose of buses being to service passenger load, the rough

similarity between shape of the bus drain and passenger load curves is unsurprising. Notable in Figure 3.5 is the symmetry of the bus levels, around the midday point, which is not matched by the passenger load; the afternoon passenger load peak is of significantly lower amplitude. Also notable in Figure 3.5 is that the supply of buses during most of the day is higher, in proportion to passenger load, than observable during parts of the morning peak. This divergence can be seen more clearly on Figure 3.6 below.

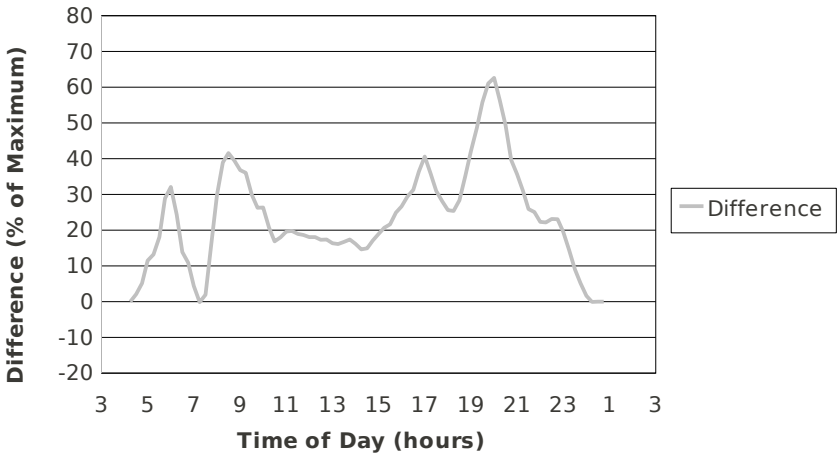


Figure 3.6: Difference Between Scaled Buses and Pax Load by Time of Day

In Figure 3.6, it can be seen that the majority of the operating time of the TransMilenio is spent at 20% over the maximally efficient point during the morning peak. At times the line runs at efficiency levels significantly lower than this, particularly during the evening peak. The efficiency with which the maximally efficient morning-peak point has been timetabled will be assessed later in this thesis, by comparing it to an ultra-efficient approach, and so, at this time, cannot be assessed. It can though, at this time, be said from the data that there appears to be potential efficiency improvements available by timetabling all periods of

the day as efficiently as the morning-peak has been timetable. It will be seen whether, further to this, the efficiency level of the morning peak can also be improved on.

This section has addressed the fourth research question of this thesis, namely whether current BRT systems are timetabled in a highly efficient fashion. This question has been answered in the negative. For the majority of the day, the Américas Line has been shown in this section to be running significantly more buses, in proportion to passenger load, than during the time of the morning peak.

3.10 Conclusion

This chapter has presented an overview of the data received from TransMilenio S.A. and how that data has been processed into a form appropriate for conducting a timetable development exercise. Particular focus has been given as to how the Américas Line bus run and passenger load data has been isolated from the rest of the TransMilenio system. The purpose of isolating the passenger load data has been to provide data with which to conduct a timetable development exercise, where as the purpose of isolating the bus run data has been to provide a comparison example for that exercise. So as to enable a realistic comparison to be made, the passenger load data has been scaled up to the rated line level of 186,000 boardings. A comparison of the bus run and passenger load data has provided an indication that efficiency gains are available to be made. With the bus run and passenger load data now in place, consideration can be given as to what might constitute ultra-efficient BRT

timetabling and how such ultra-efficient BRT timetables might be generated.

Chapter 4

An Ultra-efficient BRT Timetabling Method

4.1 Introduction

Around the world today, a great many BRT systems are being efficiently timetabled, including the test case for this thesis, Bogotá's TransMilenio BRT system. Given that efficient BRT timetabling is at least commonplace, if not the norm, what might constitute *ultra*-efficient BRT timetabling, and how might ultra-efficient BRT timetabling be clearly distinguished from today's efficient BRT timetabling? By way of an answer, this chapter will suggest a definition for ultra-efficient timetabling, one that both describes the essential nature of ultra-efficient timetabling and that also allows ultra-efficient timetabling to be clearly distinguished from today's efficient timetabling.

This chapter will begin by defining the key characteristic of ultra-efficient timetabling: the rapid tracking of passenger loads. Two non-key characteristics of ultra-efficient timetabling will also be discussed: the use of computational optimisation and the barring of intra-line transfers. Having laid out and, where necessary, argued for the characteristics of ultra-efficient timetabling, a definition for ultra-efficient timetabling will then be proposed, one that clearly distinguishes ultra-efficient timetabling from efficient timetabling. This definition will then be used to develop a method for timetabling BRT systems in an ultra-efficient manner, a manner consistent with the proposed ultra-efficient timetabling definition. Similarities will then be highlighted between the presented ultra-efficient timetabling method and today's commonplace

computational scheduling techniques, which, it will be argued, can be fairly described as being ultra-efficient scheduling techniques. Finally, this chapter will highlight and discuss critical differences between ultra-efficient scheduling and the ultra-efficient timetabling.

4.2 The Key Characteristic – Rapid Tracking of Passenger Load

It is normal for passenger loads on a transit system to vary greatly both by the time of day and between different days, such as between weekdays and weekend-days. With regard to load variation within a day, it would be normal, during an ordinary weekday, for there to be a morning inbound peak, followed by a midday lull, followed by an afternoon outbound peak, followed by a tailing-off of passenger load into the evening. As for load variation between days, there may, for example, be substantially greater load on weekdays, when people are going to work and school, than on a Saturday or Sunday.

With transit systems that do not run express services, such as most metro lines, this variation in load is dealt with simply by increasing or decreasing the frequency at which services are run. Timetabling such all-stop services is not substantially more difficult than dividing the expected passenger load by the vehicle size, the “maximum load procedure” introduced in chapter two. The planning work of getting such non-express based transit systems running well, tends not to be in the area of timetable development, but in other areas, such as passenger load estimation and vehicle and driver scheduling. For example, considerable effort was put into analysing population trends so as to assess whether to

build the South West railway line in Perth, Western Australia, and where to locate it.[140]

With transit systems which do run express services, such as the TransMilenio and other medium to high load BRT systems, the timetabling task is, by contrast, very complex. Express based transit services have to determine not only the service frequency but also the stopping patterns of the express services; just which stops should an express bus stop at? On top of having to determine the express service stopping patterns, by introducing express services, another type of service in addition to the existing all-stop service, it is also becomes necessary to determine the number of express services in relation to the number of all-stop services. For example, an operator might decide that four services are required during a given time period to service a given load, but that still leaves them to determine whether, for example, they should timetable one all-stop and three express services, or some different mix of all-stop and express services. Finally, with systems that have substantial passenger load, running multiple simultaneous express services is an option, so, in such a cases, operators would need to determine which *sets* of express services to run, and the frequency at which each service ought to to be run. For example, in a case where there were two express services plus an all-stop running simultaneously, an operator would need to determine the service frequency ratio between express service one, express service two and the all-stop service.

The TransMilenio BRT system does run multiple simultaneous express services, with the Américas Line, introduced in the last chapter,

140 Planning and Transport Research Centre (2004), *Perth's South West Metropolitan Railway: Balancing Benefits and Costs*, Planning and Transport Research Centre, Perth, Australia

running up to three express services simultaneously, in addition to an all-stop service. On the Américas Line, as of May 2005, the express services operating on the Américas Line were known as express routes 80, 100 and 120. The stopping patterns of these three express services, along with the all-stop service are shown in Figure 4.1.

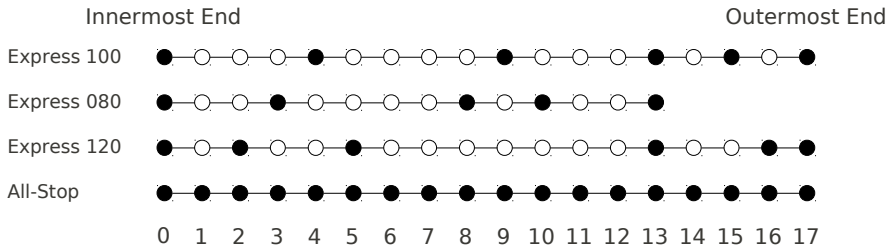


Figure 4.1: Inbound Américas Line Services

In Figure 4.1 above each stop has been numbered, with the innermost stop being assigned number 0 and the outermost stop being assigned number 17. As can be seen one of the express services, express service 80, does not travel the entire length of the line, but instead turns around at stop 13, Banderas Stop, one of the major collector stops on the line. Also of note in Figure 4.1 is the fact that the express services skip a great deal more stops than they stop at. Discounting the first and last stop of each express service, for all three express services together there are a total of 44 stops at which the express services could stop. Of these stops only 11 are being stopped at, giving a stop ratio of 25%. If the first and last stop of each express service is counted, the express services stop at 17 of a possible 50 stops, giving a stop ratio of 34%. From these ratios, it would be fair to characterise the express services on the Américas Line as infrequently stopping expresses.

In Figure 4.2 below, the times during which each of the Américas Line's three express services and one all-stop service operates is shown,

together with how frequently the service runs per 15 minutes. A 15-minute time period has been chosen here because no service frequency was in operation for less than 15 minutes, indeed only once does a service run for just 15 minutes, all other service frequencies being in operation for 30 minutes or longer.

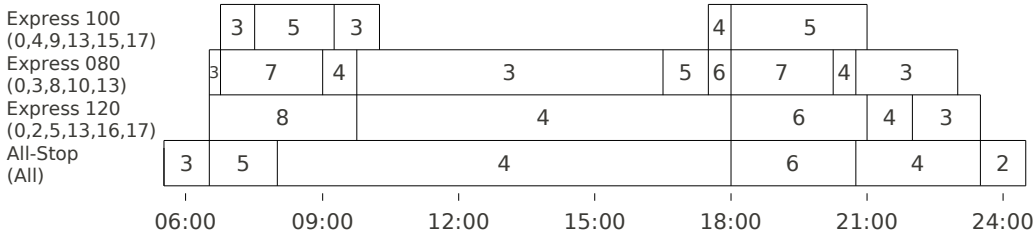


Figure 4.2: Express Service Frequency on the Américas Line

Figure 4.2 above, only shows the inbound line because the inbound and outbound services are, within the limits imposed by the need to feed buses in and out of depots and to start and stop services, essentially symmetrical. Given that, as can be seen in Figure 3.3, “Américas Line Inbound and Outbound Passenger Distance by Time of Day”, the inbound and outbound passenger load on the Américas Line is far from symmetrical, the fact that the services are essentially symmetrical is interesting; a matter that will be returned to later in this thesis.

It can be seen, in Figure 4.2 above, that service frequencies are often held constant for numerous hours. Even some of the apparent fluctuations in service frequencies are in fact transitions, over 30 minutes or so, from one long stretch of service frequencies to another. A logical diagram, which displayed these transition levels as cross-overs between two stretches rather than as distinct levels in their own right would have shown this more clearly, but such data was not available. The clearest example of service frequency stability in Figure 4.2 above is during the

midday period where exactly the same service frequency is run on all three active services, for a period of approximately five hours.

One way of looking at Figure 4.2 above would be to say that the Américas Line is clearly being serviced by a very complex set of bus services, with there being four entirely separate bus stopping patterns each of which changes its service frequency, on average, around six times a day. Compared to say a subway line that runs a single all-stop service at varying services frequencies across the day, the set of services being run on the Américas Line is indeed complicated, but compared to how complex it could be, the timetabling of services on the Américas Line is not complex; of all the multitude of stopping patterns possible on the Américas Line, only four have been used, and even though Figure 4.2 presents the service frequency in 15-minute increments, the average duration of a service frequency is numerous hours.

To illustrate this point, in Figure 4.3 below the same data as has been presented in Figure 4.2 above is shown broken up into 15-minute increments. Only the 6 am to 8 am period is shown, as this time period is sufficient to illustrate the comparative service frequency stability.

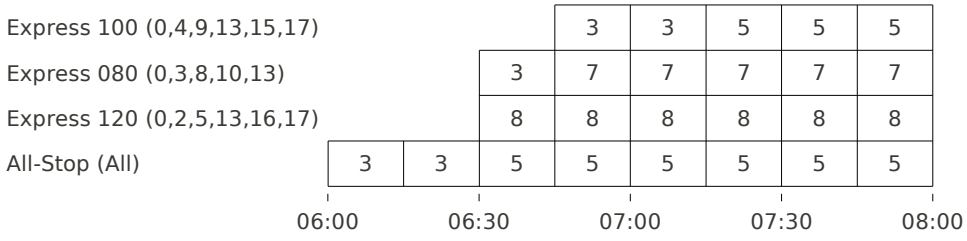


Figure 4.3: Express Service Frequency on the Américas Line in 15-minute Increments

As shown previously, in Figure 4.3 above, a 15-minute time period has been used. Although any reasonably short time period, such as 10, 20 or 30 minutes, could have been used to illustrate the TransMilenio's

comparative service frequency stability, a 15-minute time period was chosen primarily because the passenger load data received from TransMilenio S.A. was organised in 15-minute increments, and so therefore it is possible to be sure that the TransMilenio could have be timetabled down to at least this level of precision. Presented in 15-minute increments, the service frequencies of the TransMilenio appear to be changing comparatively slowly. In Figure 4.3 above, the all-stop service changes service frequency only once, from three buses per 15 minutes, to five buses per 15 minutes. As for the express services, Express 80 and 100 changes service frequency only once and Express 120 does not change service frequency at all.

In a similar way to the manner in which the service frequencies have been ‘unpacked’ in Figure 4.3 above, to show their comparative stability, an ‘unpacking’ process can be conducted for service stopping patterns. In Figure 4.3 above, the service stopping patterns were shown on the left hand side of the graph, by convention indicating them to be the same across each of the four rows of services. In Figure 4.4 below, this relationship between the stopping patterns and the 15-minute service frequency time blocks, has been made explicit, by moving the stopping patterns inside the 15-minute time blocks.

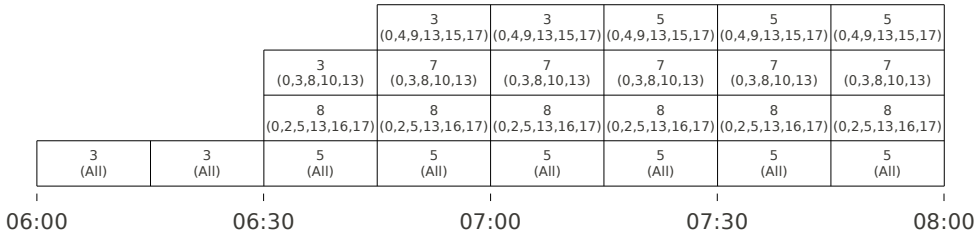


Figure 4.4: Express Service Frequency and Stopping Pattens on the Américas Line in 15-minute Increments

By showing in place both the service frequencies and stopping patterns of the Américas Line timetable, during the 6 am to 8 am time period, the relative stability of Américas Line's timetable is clearly discernible. The line has been timetabled with service frequencies that change relatively slowly and with stopping patterns that do not change at all. Of the multitude of possible stopping patterns four have been chosen, the all-stop pattern and three express patterns. Beyond this, the only extent to which the Américas Line's four stopping patterns exhibit variability, is that each of the four selected stopping patterns is sometimes in operation and sometimes not in operation. Passenger load is nowhere near as stable as the timetable above, as could be seen in Figure 3.5, "Buses and Pax Load Compared by Time of Day."

To give an idea of just how varied the service frequencies and stopping patterns on the Américas Line might have been, and how this might look compared to Figure 4.4 above, Figure 4.5 below shows the same set of 15-minute 'boxes' but with the boxes having been filled with highly variable dummy data. This dummy data has been constructed to look realistic, but is otherwise completely random.

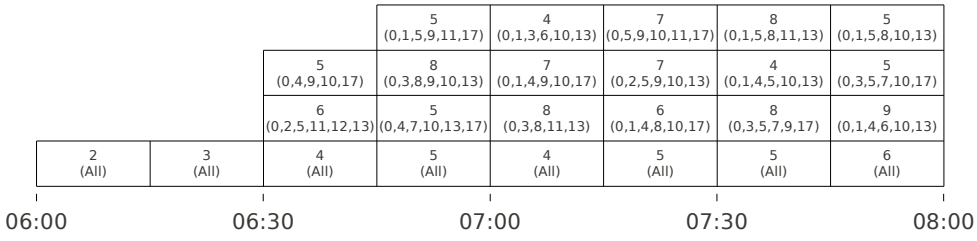


Figure 4.5: Express Service Frequency and Stopping Pattens using Dummy Data in 15-minute Increments

It is the primary task of this thesis to examine the question of whether a timetable that looks like Figure 4.5 above is significantly more efficient than one that looks like the existing timetable, as shown in

Figure 4.4. This thesis will, as shown above, timetable services down to the level of 15-minute time slots, with this 15-minute level having been chosen because the passenger load data was received from TransMilenio S.A. in 15-minute increments and, as this thesis is about ultra-efficient timetabling, there seemed no reason to coarsen this granularity. The question will be: can a timetable that varies its service frequencies and stopping patterns at the same rapid pace that passenger load levels and patterns change, significantly outperform a timetable that does not exhibit quite such ‘hyperactive’ variability?

4.3 A Practical Characteristic – Computational Optimisation

Irrespective of how slowly or rapidly a timetable tracks passenger load, a method is needed to compute a set of bus services for a given passenger load matrix. In the case of a timetable that slowly tracks passenger load, the peak passenger load during a time period might be used to represent the load matrix for the whole time period, so as not to under-service a line. For example, if a set of bus services is being computed for a one-hour time period and there are four 15-minute passenger load matrices available, the one with the highest load would commonly be used. Although, in this thesis each 15-minute time period would be timetabled separately, producing four different service patterns, the problem is essentially the same: how do you produce a service pattern from a passenger load matrix? Two approaches to this problem could be used: a manual or at least semi-manual approach, and an automated approach.

TransMilenio S.A. use a semi-manual approach to produce a service pattern from a passenger load matrix.[141] Their strategy is to construct an express service out of several high-load stops, and then to 'tune' this express service by adding other smaller stops or groups of smaller stops in an attempt to create a service that has high occupancy levels along the length of the line. A spreadsheet is used to automate the collation of load data, but otherwise the process of producing service patterns from passenger load matrices is done 'by hand'; a highly time consuming process. In considering whether such a semi-manual approach might be realistically applied to produce service patterns for an ultra-efficient timetable, the scale of the work required needs to be assessed.

On the TransMilenio system as a whole there are four times the number of services as on the Américas Line alone, 16 services for the system as a whole as compared to four for the Américas Line. It would appear, though, from Figure 4.2, "Express Service Frequency on the Américas Line", that service patterns for the existing efficient timetable are being held, on average, for at least an hour, as opposed to the 15-minute time duration which will be used for the ultra-efficient timetabling exercise in this thesis. So, with the TransMilenio system as a whole having four times the number of services as the Américas Line, but with those services changing their frequency at one-quarter the pace, the number of service changes that will be timetabled in this thesis, will be similar to the number of service changes on the entire TransMilenio system.

141 From notes taken at a meeting between the author and Alejandro Niño of the TransMilenio S.A. Planning Department on the 7th of June 2006.

Furthermore, in terms of assessing the comparative computational complexity, the service changes metric just computed would provide a minimum estimate, as it does not take into account the fact that when the TransMilenio system changes services it selects from a small pallet of expresses, 12, whereas the timetabling to be conducted in this thesis will select new expresses for every 15-minute time period. TransMilenio S.A. has a department to do their timetabling. Even giving due consideration to the fact that that department has to concern itself with practical deployment issues, whereas this thesis does not, attempting to replicate the planning department's effort does not seem a realistically attainable goal. Consequently, due primarily to practical concerns, a fully-automated approach would seem a better choice to produce the service patterns for an ultra-efficient BRT timetable.

Along with this practical concern, that if a manual optimisation method were used then excessive manual resources would be required, it also seems appropriate, for a thesis entitled Ultra-efficient Bus Rapid Transit Timetabling, to develop at least a rudimentary method of automated optimisation. In considering how to automate the production of service patterns for an ultra-efficient BRT timetable, the first question to arise is whether, for the Américas Line test case in this thesis, the problem is intractable. In chapter two the fact that some combinatorial problems cannot be optimality solve in a realistic amount of time was discussed, with such problems being referred to as intractable. One of the reasons for choosing the Américas Line as the test case for this thesis, was that, as a relatively short line, it was less likely be an intractable computational optimisation case.

In chapter two it was shown that for an example line of 27 stops, that has two express services running, there are approximately one quadrillion possible express service patterns for a given passenger load matrix. The same method used to show that this ‘normal’ transit line can be timetabled in a vast number of different ways, can be used to assess the number of different ways the real-world test case for this thesis, the Américas Line, can be timetabled. The Américas Line has up to three simultaneous express services running, in addition to a constantly running all-stop service. If, as in the previous example in chapter two, it is assumed that all express buses must stop at the first and last stop on the line, as they are turn-around stops, that leaves 16 middle stops along the line where each of the three expresses might be timetabled to stop at, or not. Combining these three express services, with 16 middle stops each, gives 48 stops that can be timetabled to be stopped at, or not. With two choices, stop or not-stop, and 48 stop where that choice can occur, the number of timetabling options for this situation is two to the power of 48, or 281,474,976,710,656 timetabling options. Given an appropriate metric to access efficiency, one of these options out of the approximately 281 trillion possible timetabling choices will be the most efficient.

So the Américas Line case is not quite as difficult as the general example case given in chapter two with ‘only’ approximately one-quarter of a quadrillion possible timetabling options. As will be shown in the next chapter, for each of these timetabling options extensive calculations need to be made to construct a metric by which the timetable can be assessed. A rough preliminary time estimate was made of how long it would take to generate a full-days optimal timetable for the Américas Line on a normal

computer,[142] which yielding an figure of approximately 320 years.[143] Although this figure does not make the problem of generating a full-days optimal timetable for the Américas Line intractable in an absolute sense, the problem could be given to a super-computer, in the context of the resources available for a Ph.D. thesis, the problem is clearly intractable. With an optimal solution being effectively unavailable, the question therefore is what method to use to produce a suboptimal solution, a good rather than perfect solution.

One approach would be to attempt to apply the sophisticated techniques used to produce transit schedules, that were briefly discussed in chapter two, to the problem of timetabling. The problem with applying these sophisticated techniques is just that, they are sophisticated, and far from represent the quickest way of obtaining a ‘good’ solution to the problem in question. As was discussed in section 1.3, “Scope and Approach of the Research”, due to the decision to both use real-world data and to build a real prototype software tool, a decision that greatly complicated the research, simple methods were to be favoured elsewhere whenever possible. Consequently, the simplest approach to producing

142 The virtualised computer used had a Intel Core 2 Duo 2 Ghz CPU with 384 megabytes of memory and was running Ubuntu Linux 9.0.4. Virtualisation was provided by VirtualBox 2.2.4. The second processor core was not directly used as the software developed here was not designed with multi-core support.

143 To generate an optimal timetable the cases where there are three express services running would require two to the power of 48 (281,474,976,710,656) stopping options to be assessed. The software developed here was able to assess two to the power of 16 (65,536) stopping options 136 times in 40 seconds, or at a rate of one stopping pattern every 0.000004488 seconds. At this rate it would take approximately 40 years to optimally assess one case of two to the power of 48 stopping options. In the final timetable developed here, shown in appendix two, there are eight cases where three expresses are used, so approximately 320 years would be required to optimally assess these eight cases.

‘good’ express service patterns was tried first to see if it yielded results worth reporting.

The simplest approach to producing ‘good’ express service patterns is to sequence the assessment of express services, rather than to calculate them simultaneously. So rather than considering express services one, two and three at the same time, yielding the previously mentioned one-quarter of a quadrillion possible timetabling options, service patterns are calculated first for express one, and then for express two, and then for express three. By sequencing the calculations optimality is lost, but also the number of timetabling options that need to be assessed is reduced from two to the power of 48 to a level of two to the power of 16 times three, a mere 196,608 options. Preliminary results strongly indicated that assessing this number of timetabling options was both well within the computational capability of a normal computer, and yielded worthwhile results, so this was the approach used. A detailed description of exactly how the assessment of express services has been sequenced is given in the next chapter.

This section has addressed the fifth research question of this thesis, namely whether it is possible to generate optimally efficient timetables for BRT systems. For BRT systems which run express services, side-by-side with an all-stop service, this section has shown that it is not always possible to generate an optimally efficient timetable in a realistic period of time.

4.4 A Theoretical Characteristic – No Intra-line Transfers

In the previous two sections the issue of matching service patterns to passenger load matrices was discussed, first with regards to how rapidly load is to be tracked and then with regards to how precisely services are to be mapped to load. Neither of these sections, though, have addressed the issue of potential interplay between loads and services, beyond oblique references to “practical deployment issues.” There are two ways in which these factors can interact.

Firstly, as with the price of a product in economic theory, passenger load is not a constant but is the outcome of interplay between supply and demand. On a single isolated transit line, this interplay would be difficult to analyse; with the reality being that most transit line are a part of transit networks, the situation complicates further. Dr. Paul Mees discusses the effect of transit lines being a part of a transit network in his book *A Very Public Solution: Transport in the Dispersed City*, noting that if transit lines are integrated together into a transport network grid, passenger load (patronage) can increase markedly.[144] The designers of the TransMilenio BRT appear to be well aware of this effect, known as the network effect, with the final state of the TransMilenio system, slated for 2016, being a grid of routes, as shown in Figure 4.6 below.

144 Mees, P. (2000), *A Very Public Solution: Transport in the Dispersed City*, Melbourne University Press, Victoria, Australia, pp. 138-142

Figure 4.6: Expected Coverage of the TransMilenio BRT in 2016[145]

Once the context of a transit line, where it fits within the network it is a part of, has been taken into account, numerous supply and demand issues will determine the load on that line. For example, if more people move to live close to a transit line there will be a higher demand for mobility and consequently load will go up. Conversely, if the frequency of services on a transit line were reduced there would be a lower supply of mobility services, in this case via a quality reduction, and therefore load would fall. For the purposes of this thesis this first type of load/service interaction has not been taken into consideration. The total passenger load has been treated as a constant.

The second type of load/service interaction occurs on express based transit systems, where passengers are able to switch between all-stop and express services to reach their destination. An example of switching between services, would be a person who travels between stop A and stop B by boarding an all-stop bus from stop A to an in-between stop C and then switching to an express bus to travel from in-between stop C to their destination stop B. If passengers switch between buses then their

145 Grutter Consulting, *BRT Bogotá, Colombia: TransMilenio Phase II-IV: Project Design Document*, TransMilenio S.A., Bogotá, Colombia, p. 21

trip, which, in this example, was initially between stop A and stop B is now, in effect, between stop A and stop C and then between stop C and stop B. How people make this choice of whether or not to switch from an all-stop to an express service is a complex matter. A paper called *Passenger Utilization of Local vs Express Trains for a New York City Subway Line* examines this matter, and notes that people's choices are not always 'rational' in that "passengers consistently overestimated their travel time and correlated their use of faster trains with faster service."[146] The switching of passengers between all-stop and express services is important because, although it does not alter the overall load on a line, it does change the load *pattern*.

A change in the passenger load pattern is a problem if the bus routes that are servicing that load pattern have been determined on the basis, continuing with the example above, that there is a one-person load between stops A and B. A one-person load between stops A and B is not the same as a one-person load between stops A and C and then a one-person load between stops C and B. In theory one could adjust the bus patterns to take into account this switching between buses, but in practice people switch between all-stop and express buses on the basis of the bus patterns, and so a circular relationship exists between the bus patterns and the passenger load patterns; the bus patterns are to a degree determined by the passenger load patterns and the passenger load patterns are to a degree determined by the bus patterns.

As with the question of efficiently matching a service pattern to a load matrix, resolving this circularity would be a topic fitting of a thesis

146 Wiener, R., Lidor, G. (1978), *Passenger Utilization of Local vs Express Trains for a New York City Subway Line: A Case Study*, Transportation Research Record, Volume 662

itself. The circularity in question could be stabilised by finding a load-pattern/service-pattern pair that were matched such that each did not vary the other. It is likely that the computational tool-set required to carry out such a stabilisation, in an automated manner, would be highly sophisticated, and no such automated tool-set was found in the literature.

It is the case though, that if passengers are allowed to transfer between services, achieving this stabilisation, in some manner, is not optional. TransMilenio S.A. achieves their timetable stabilisation by feeding their preliminary timetables into EMME/2, to model actual passenger behaviour, and then modifying these timetables where EMME/2 has shown there to be significant weaknesses, such as buses going over-capacity or buses travelling greatly under-capacity. This process is repeated until no further significant weaknesses are present. This approach could at best be described as semi-automated, but would perhaps be better described as a manual approach which makes use of a powerful generic computational tool in the form of EMME/2.

This thesis, in contrast, will address this circularity issue by disallowing transfers. As was done in the previous section, an argument could be made for this position of disallowing transfers on the basis of practical resource availability. Feeding data back and forth between a timetabling system and EMME/2 until it stabilises, is a task for a planning department not a person. A better argument for disallowing transfers though is a theoretical one, specifically that a strong efficiency case can be made against intra-line transfers.

Letting passengers switch between all-stop and express buses is inherently inefficient as every transfer doubles that passenger’s board

and alight time. Although it might be possible to manage the interaction between passenger intra-line transfer practices and bus service patterns, so that the passenger intra-line transfers practices were leveraged to consciously form bus service patterns that were sufficiently efficient to more than compensate for the added board and alight time, the sophistication required to do so is far in excess of currently available techniques. Consequently, if an ultra-efficient outcome is the goal, then at this time, disallowing intra-line transfers is an appropriate approach, and the one that will be followed in this thesis.

4.5 Defining Ultra-efficient Timetabling

The preceding sections have discussed the characteristics of ultra-efficient timetabling, as it is to be conducted in this thesis. The primary characteristic of ultra-efficient timetabling, the rapid tracking of passenger loads, has been discussed and two secondary characteristics, computational optimisation and the barring of intra-line transfers, have also been considered. Discussing the characteristics of ultra-efficient timetabling is necessary if a method is to be developed to carry out ultra-efficient timetabling, but together these characteristic do not constitute a definition of ultra-efficient timetabling. Before moving on to look at exactly how these characteristics might be formed into an ultra-efficient timetabling method, a definition of what ultra-efficient BRT timetabling is will be developed in this section.

The raison d’être of a definition is to provide a way to distinguish a concept or a thing, in particular a way to distinguish one concept or thing from another concept or thing, especially when the two concepts or

things that are being distinguished from one another appear superficially similar. There is already a concept called efficient BRT timetabling, it is practised around the world. For there to be a clear separate category of timetabling, called ultra-efficient timetabling, there must be a clear difference between ultra-efficient timetabling and efficient timetabling, otherwise ultra-efficient timetabling would rightly be considered to be simply a more efficient form of efficient timetabling, not a separate category of timetabling. The primary job of an ultra-efficient timetabling definition is therefore to provide a clear threshold test; a threshold which ultra-efficient timetabling clearly meets and that efficient timetabling clearly does not meet.

An obvious point from which to develop a definition for ultra-efficient timetabling is the primary ultra-efficient timetabling characteristic, given at the start of this chapter, that is the rapid tracking of passenger loads. If ultra-efficient timetables are those that are said to rapidly track load, then for such timetables to be different from efficient timetables, efficient timetables must be said to not rapidly track load. From this characteristic it is possible to specify a threshold test for ultra-efficient timetabling, by providing an unambiguous definition of “rapid.”

It would be entirely possible to devise an unambiguous quantitative definition for the concept “rapid.” For example, it could be said that passenger load is being rapidly tracked if express services change their service pattern or frequency at least, on average, every 30 minutes. The TransMilenio BRT does not track load this rapidly and so it could be said to not to be timetabled in an ultra-efficient manner, whereas the timetable development to be conducted here will be based on 15-minute

time slots and so could be said to be timetabled in an ultra-efficient manner. Although unambiguous, this definition of ultra-efficient timetabling does beg the question of why the cut-off is set at 30 minutes, why not 29 minutes or 31 minutes. However carefully this numerical level was refined there would always be the difficult to answer question of why not one minute more or one minute less than any chosen time level.

A qualitative definition is the alternative. This thesis stemmed from a previous non-BRT timetable development project[147] that, due to an attempt to deploy very complex timetables, had as a basic assumption the use of a fully-automated passenger information system. Fully-automated here refers to a passenger information system where passengers have no real involvement in selecting their route, they simply arrive at their departure stop and a live passenger information display board tells them which vehicle they need to board. This type of fully-automated passenger information system differs somewhat from the common situation, where the aim is to keep passengers informed of the state of a transit system in real time. Such real time information (RTI) systems are researched by industry groups such as the UK Real Time Information Group, who produce papers such as *The Integration of RTI into Bus Fleet Management*[148] and *RTIG Monitoring: Enablers and Blockers to Rollout of RTI Systems*.[149] Such RTI systems are not directly comparable to a real time information system informing people which bus they should board,

147 Bradley, M. (2003), *Flexways: The Advantages of a Tactical Imbalance in Transport Planning* (Honours Thesis), Murdoch University, Perth

148 UK Real Time Information Group (2005), *The Integration of RTI into Bus Fleet Management*, UK Real Time Information Group, Guildford, United Kingdom

149 UK Real Time Information Group (2005), *RTIG Monitoring: Enablers and Blockers to Rollout of RTI Systems*, UK Real Time Information Group, Guildford, United Kingdom

but it is the case that real time information systems are, at minimum, a rapidly maturing technology.

Although little else has carried through from the previously mentioned non-BRT timetable development project to the research of this thesis, the idea of assuming a fully-automated passenger information system has. To some degree the genesis of the idea to see just how precisely BRT systems could be timetabled stemmed from the idea of having fully-automated passenger information systems. In melding the idea of a fully-automated passenger information system with BRT, the question arose that if, at least in theory, a timetable of any complexity could be deployed, how much more efficiently would BRT systems be able to be run.

By combining the key characteristic of ultra-efficient timetabling, the rapid tracking of passenger load, with the conceptual genesis point of this thesis, fully-automated passenger information systems, it is possible to give a qualitative definition for ultra-efficient timetabling. A BRT system can be said to have been timetabled in an ultra-efficient manner if:

> its services change so rapidly that passengers can no longer be reasonably expected to determine by themselves which service they need to board.

So, when paper timetables become infeasible, when passenger load is being tracked so rapidly that a fully-automated passenger information system is the only realistic way for passengers to determine which

service they need to board, then that service can be said to have been timetabled in an ultra-efficient manner. By this definition the TransMilenio is not timetabled in an ultra-efficient manner, the TransMilenio has always used paper timetables.[150]

Sufficient parameters of the ultra-efficient timetabling method to be presented in this thesis have been defined to give a rough estimate of how large an ultra-efficient timetable would have to be if it were to be presented on paper. As can be seen on Figure 4.2, “Express Service Frequency on the Américas Line”, the Américas Line is open for approximately 19 hours a day (05:30 to 24:30) and on average is running two express services at any given time. If this line were to be timetabled using these basic parameters, but in an ultra-efficient manner, there would be two different express service for each 15-minute period, giving 152 different express patterns per direction, or 304 different express patterns for the line in total. A recent TransMilenio timetable[151] that details the stopping pattern for each express, presents two express stopping patterns in a 9 cm by 30 cm space, so using approximately 135 square centimetres per express stopping patterns. Presenting the estimated 304 different express patterns for the Américas Line alone, would take approximately 4.1 square metres of paper. If the same area were needed for the TransMilenio’s other two physical lines then a total of 12.3 square metres of paper would be required to present the express service information.

150 Current TransMilenio timetables available from: *Guía General de Servicios*, TransMilenio S.A. Website, Available: http://www.transmilenio.gov.co/WebSite/Contenido.aspx?ID=GuiaGeneralDeServicios (Accessed 19 February 2009)

151 *Servicios Expresos: 17 - 18*, TransMilenio S.A. Website, Available: http://www.transmilenio.gov.co/AdmContenidoUpload/administrador.contenido/Files/GuiaGeneral/nvoguia_17-18.pdf (Accessed 19 February 2009)

To convey the scale of such a timetable using a well known format, if 12.3 square metres of information were to be packaged into a b-format paperback book,[152] the size commonly used for works of literature, which is approximately 20 cm high by 12.5 cm wide, the book would have to be 492 pages long; the size of a substantial novel. By comparison, a similar presentation of the 12 express services on the TransMilenio system, as at May 2005, would require b-format book approximately 6 pages long; approximately one-eightieth the size. Such calculations are indicative only, but they do convey the extent of the difference between an efficiently timetabled BRT service and one that has been timetabled in an ultra-efficient manner. For a computer database the quantity of information printed in a novel is minuscule, for a person a book-length timetable would at least be least be a foreign experience and would at worst be totally unacceptable. Perhaps if transit systems had had a different historical development path, the idea of deploying a book-length timetable might be feasible, people do after all use telephone books which are often much larger. Given the actual situation, though, where people expect to be able to understand how their transit systems work preferably at a glance, deploying a book-length timetable is not realistic.

This section has addressed the sixth research question of this thesis, namely whether it is possible to develop a definition of ultra-efficient BRT timetabling, one that clearly differentiates it from current BRT timetabling. This question has been answered in the positive, with a clear qualitative definition for ultra-efficient BRT timetabling having been developed and presented in this section.

152 *The Importance of Format and Feel* (2001), The Guardian Website, Available: http://www.guardian.co.uk/books/2001/aug/11/gettingpublished (Accessed 23 January 2009)

4.6 An Overview of an Ultra-efficient Timetabling Method

Knowing what ultra-efficient timetabling is, does not mean one knows how to timetable in an ultra-efficient manner; a definition is not a method. A definition allows one thing to be distinguished from another, a method is, in this context, a set of steps which can be followed to solve a problem. The ultra-efficient timetabling ‘problem’ is non-trivial and so the solution outlined in this section, although not very complex, will be non-trivial.

In describing any reasonably complex solution, the issue arises of whether to describe the components of the solution first, or, alternatively, to start by presenting an overview of the solution as a whole. If one starts by describing the components, then this has to be done without an overall context. Conversely, if one starts by describing an overview to the solution, then this has to be done without an understanding of why particular components to the solution have been chosen, or the justification for the selection of those components. Although there is no ideal way of presenting complex models, presenting a general overview first tends to be the better of the two available options.

This section will present a general overview of the approach used to do ultra-efficient timetabling in this thesis, without going into actual execution details. The basic approach that will be used to timetable the Américas Line in an ultra-efficient manner will be:

1. Treat each 15-minute period on the inbound and outbound line entirely separately.

2. During the time that the line is open, run a minimum number of all-stop buses for every 15-minute period, regardless of whether there is the load to justify those buses, so as to provide a guaranteed minimum level of service.

3. For each 15-minute period, search a very large number of service patterns, and pick the one that most efficiently matches the passenger load pattern. This may involve running all-stop services more frequently than at the minimum level, but may not involve running them less frequently.

4. Use deadheading[153] to balance up the bus flows.

This method is, in very general terms, how the process of ultra-efficient timetabling will be conducted by the software package to be developed for this thesis. Specifically how this method translates into software can be seen in Figure 4.7 below, which shows the data and processing elements of the software as a data-flow diagram, with the data elements shown as rectangles and the processing elements shown as ovals. The naming of functions in Figure 4.7 is in lower case, in line with a widely employed computer science convention.

153 Deadheading means running a bus from one place to another without passengers and without stopping at any stops.

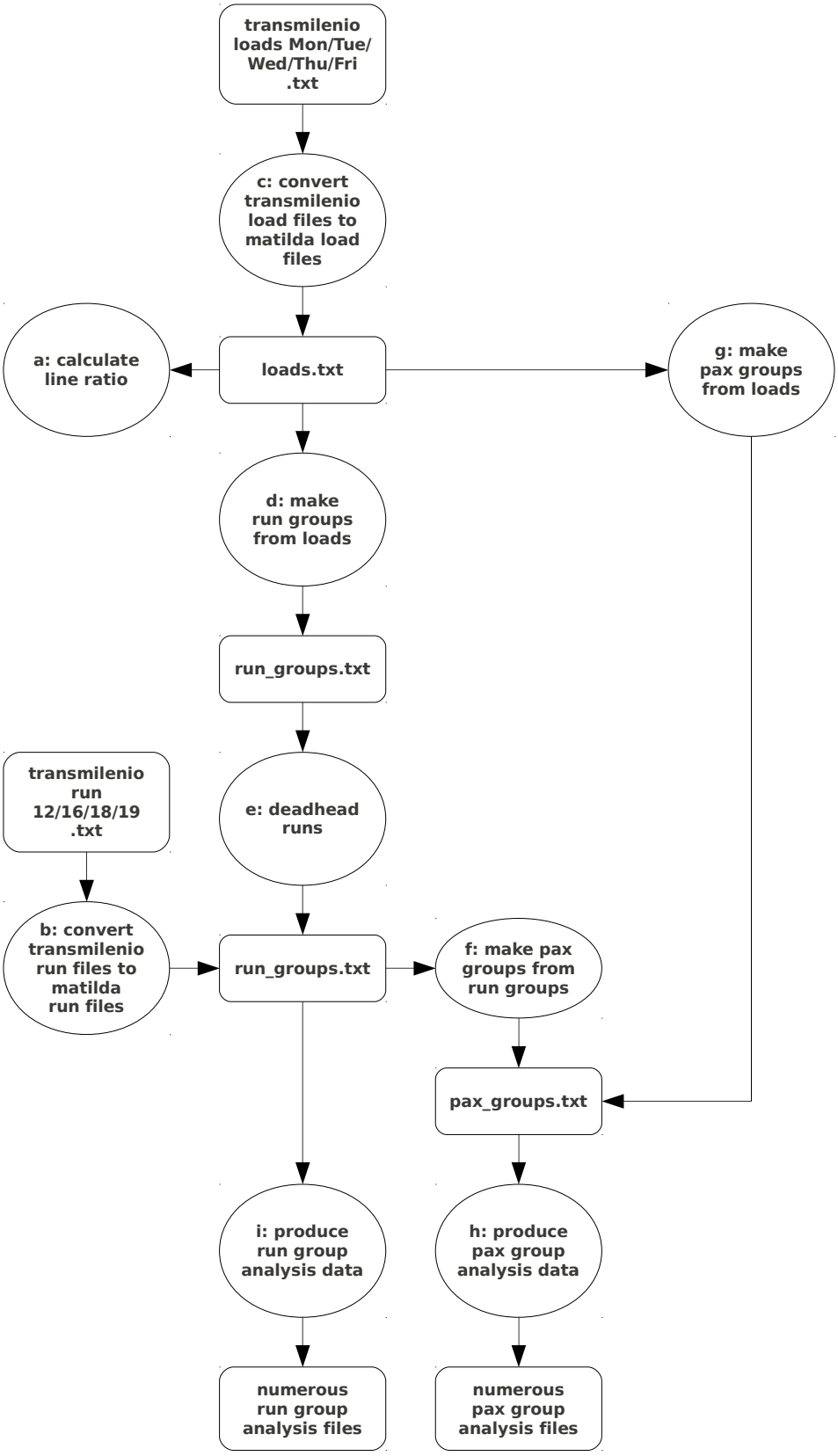


Figure 4.7: Data-flow Diagram for the Ultra-efficient Timetabling Software

Although Figure 4.7 shows a number of processing pathways, all of these pathways are minor except for the primary pathway, the one which produces the ultra-efficient timetable. The pathway that produces the ultra-efficient timetable begins at the top of Figure 4.7 with the TransMilenio load files, called “transmilenio loads Mon/Tue/ Wed/Thu/Fri.txt.” These five files, one for each day of the working week are not suitable for constructing an ultra-efficient timetable and so are converted by function “c”, “convert transmilenio load files to matilda load files”, into a single file called “loads.txt.” Function “d”, “make run groups from loads” reads in this file to generate the ultra-efficient timetable run groups and writes out these run groups in the file “run_groups.txt.” Function “e”, “deadhead runs” reads in the “run_groups.txt” file adds deadhead runs to the run groups sufficient to balance up the runs and then re-outputs the “run_groups.txt” file. At this point the main processing pathway splits into a left and right fork. On the left fork, the data in the “run_groups.txt” file is converted by function “i”, “produce run group analysis data”, into numerous run group analysis files. On the right fork, the “run_groups.txt” file is converted by function “f”, “make pax groups from run groups” into passenger groups, and then outputted into the file “pax_groups.txt.” This conversion is possible because the function “make run groups from loads” which originally makes the “run_groups.txt” file, records the passengers associated with each run. Finally, the “pax_groups.txt” file is converted by function “h”, “produce pax group analysis data”, into numerous pax group analysis files.

An addition to this primary processing pathway, there are three minor pathways, one to generate a line ratio, whose definition and

purpose will be discussed in the next chapter, and two to produce passenger and run analysis data for the existing efficient timetable. In Figure 4.7, the pathway that produces the line ratio can be seen as a branch to the left of the “loads.txt” file, with function “a”, “calculate line ratio”, processing the “loads.txt” file into a single needed line ratio value which is outputted to a terminal window. The pathway that is producing the run analysis data for the existing efficient timetable can be seen starting on the left hand side of Figure 4.7 with the TransMilenio run files, called “transmilenio run 12/16/18/19.txt.” These four files, one for each of the four services on the Américas Line, are converted by function “b”, “convert transmilenio run files to matilda run files” into a single file called “run_groups.txt.” In the same manner as was detailed in the last paragraph, this file is then fed down the left hand fork of the primary pathway to produce numerous run group analysis files. Finally, so as to be able to produce passenger analysis data for the existing efficient timetable, function “g”, “make pax groups from loads”, to the right of the “loads.txt” file, takes this file and converts it into a “pax_groups.txt” file. This file is then fed down the right hand fork of the primary pathway to produce numerous passenger group analysis files. Details as to how all of the processing steps shown in Figure 4.7 operate, will be given in the next chapter.

Although there are many steps in generating an ultra-efficient timetable, the most important of the above cluster of tasks is function “d”, “make run groups from loads”, which determines which runs will be selected to service a given passenger load. At the centre of this function a metric will be needed to judge whether or not one candidate run group

is better than another. As this thesis' focus is on the efficient operation of BRT systems, not the provision of a rapid passenger service, the development of a metric will be similarly focused. Consequently, the chosen metric is minimum total bus run time, the total time that all the buses for a given time period are out on the line. So, although passenger transit speed has not been directly catered for, this metric of, in effect, having the buses run as quickly as possible has a clearly positive tendency towards passenger travel times.

As discussed in section 4.3, "A Practical Characteristic – Computational Optimisation", function "d" will cycle through different express service stopping patterns, searching for an efficient solution. It is in this context that the metric of minimum total bus run time will be applied to judge whether one given express service stopping pattern is better than another. The optimisation method to be used will conduct an exhaustive search of all possible timetabling options for the one express service case. Consequently, for this case, the timetabling outcomes will be optimal, as assessed by the metric used.

By contrast, in the two and three express service cases, only a small subset of all possible timetabling options will be assessed, and so it would be expected that the resulting timetable outcomes will be sub-optimal. The production of sub-optimal timetables is not a problem, as such, as the stated aim of this thesis is to try and produce timetables that are better than existing efficient timetables, not to produce optimal timetables. If, though, the optimisation method used were to produce highly sub-optimal outcomes for the two and three express service cases, that would

compromise the objective of trying to produce timetables that are better than existing efficient timetables.

When searching only a subset of a large combinatorial solution space, it is known that, in certain cases, highly inefficient solutions can result. This problem is sometimes known as the problem of being caught in a sub-optima; a situation where the optimisation metric has got ‘stuck’ on a solution which is far from optimal, leaving a large number of more fruitful solutions un-assessed.

The hill climbing optimisation method is an example of an optimisation method where there is a risk of being caught in a sub-optima. Hill climbing involves examining the options around a given potential solution and moving towards better solutions; climbing up the hill, as it were, to the ‘peak of the hill’ which represents the optimum solution. This approach works fine so long as the hill in question has only one peak. If, though, the hill were to have two peaks, one lower than the other, there is a real possibility that the hill climbing approach will get ‘stuck’ at the lower peak, because a peak, any peak, is by definition a point where every surrounding point is at a lower level. In this example, the lower peak would be referred to as a local optima, as it is the optimum solution locally, even though it may be highly sub-optimum globally.

The optimisation method detailed in this section sequentially produces the needed number of express services for the two and three express service case. At each stage of the production sequence the outcome will be optimal, as all of the possible 65,536 express patterns are assessed. Between the stages though, as earlier stages hand over to

later stages, there is a real risk of being caught in a sub-optimal solution. Each stage generates express patterns by drawing from the same ‘pool’ of passengers, with earlier stages of the optimisation sequence getting to take the passenger loads they want before later stages have access to those loads. Consequently, there is a risk that an early stage of the optimisation sequence may select an express pattern which makes it impossible for a later stage to find its own efficient express pattern. This situation has the potential to yield a highly sub-optimal solution.

How this risk of being caught in a sub-optimum has been managed is addressed in the next chapter. Other important issues arising from the ultra-efficient timetabling method given in this section will also be discussed in the next chapter, including the rationale behind the use of deadheading and the method used to achieve efficient deadheading.

4.7 Similarities to Ultra-efficient Transit Scheduling

Highly computationally intensive methods which produce ultra-complex outputs, ones which no human would ever have directly produced, are already widely used in the transit sector, in the area of vehicle and driver scheduling. Recall from chapter two that vehicle scheduling is the process of assigning a particular vehicle in a transit fleet to each of the timetabled trips and joining up the trips such that they begin and end at depots, and that driver scheduling is the process of assigning a particular driver in a transit-driver-crew to each of the timetabled trips. As discussed in chapter two, the organisation of bus and driver schedules by scheduling software, such as that produced by Giro Inc., is highly computationally intensive, and although a great deal of

effort has been invested in making the output bus and driver schedules comprehensible, this does not alter the fact that these outputs are ultra-complex.

To manage this complexity, it is normal for scheduling software to have components to help operators organise driver schedules. For example, the Giro company's Hastus scheduling system has the component Hastus - Roster to help operators "prepare efficient weekly or periodic crew assignments" and Hastus - Crew to helps operators "build efficient timetables to cover vehicle schedules."[154] Other scheduling systems have similar components, such as the Omnibus scheduling system with its OmniROTA and OmniDRIVER components.[155]

It is worth considering why computationally intensive transit scheduling produces complex output and whether, as a matter of first principles, ultra-efficient timetabling is likely to produce similarly complex output; is a complex timetabling method likely to identify a simple timetable as 'optimally' efficient, and if not why not?

It is not necessarily the case that more complex timetables are more efficient than simple ones, it is only highly likely for this to be the case. When, in the case of transit scheduling, a computer algorithm is faced with billions or trillions of possible scheduling options, it will be the case that only a very small subset of these options map onto some readily human-recognisable pattern, and consequently can be referred to as simple. Such a 'simple' pattern may be, in the case of scheduling, that within a system any given bus only services a given route on a given day.

154 ibid.

155 *OmniROTA*, Omnibus Company Website, Available: http://www.omnibus.uk.com/omnirota.html (Accessed 7 February 2009)

Such a scheduling arrangement is so simple it can be clearly described in a single sentence. To people this scheduling patterns is special, it is a type of special we call simple. To a computation optimisation routine this example of a simple schedule is not special, it is just one way of arranging things amongst a sea of options. Unless an optimisation system is specially structured to favour such a ‘simple’ option, or the system it uses to judge options is calibrated so as to recognise and favour them, then such simple options will be just one of the often billions or trillions of options, ‘competing’ to be the most efficient. It is possible, as a matter of chance, that a ‘simple’ option might be the most efficient, but it is so unlikely that we fairly characterise the output of computationally intensive scheduling systems as complex.

As was discussed in chapter two, ultra-efficient timetabling has a great deal in common with normal transit scheduling, in that both methods involve searching through very large solution spaces for efficient solutions. It might be with ultra-efficient timetabling that a simple timetable emerges at the end of such process, but, as the situation is structurally the same as for normal scheduling, ultra-efficient timetabling is likely to share normal scheduling’s ultra-complex output; a significant similarity.

4.8 Differences from Ultra-efficient Transit Scheduling

Although the character of the information resulting from ultra-efficient timetabling will be ultra-complex, in the same way as for normal scheduling, there are import differences between normal scheduling and ultra-efficient timetabling relating to who uses or ‘consumes’ that

information. Normal scheduling produces ultra-complex bus and driver schedules, which are ‘consumed’ by drivers, whereas ultra-efficient timetabling produces ultra-complex timetables, which are primarily ‘consumed’ by passengers.

Passengers and transit drivers are quite different classes of people, in at least two important ways. Firstly, passengers are the general public, exhibiting the full spectrum of temperaments, intelligence levels and other such characteristics. Transit drivers, by comparison, are a small subset of the general public, carefully selected, one would hope, to exhibit those characteristics best suited to their role. Secondly, although “transit driver” and “passenger” are just roles, and people can and do regularly go from being a transit driver to a passenger and visa versa, in the current socio-economic milieu it is seen as appropriate that drivers, as employees, be compliant, whereas passengers, as consumers, be demanding.

Furthermore, drivers are not driving because they wish to go somewhere; they are helping to provide a given transit systems’ mobility services, not consuming those services. Consequently, a driver’s ‘travel’ for a given day can, on the whole, be set at the beginning of that day, if not before, and a custom schedule produced for them that details just their movements for that day. In this manner, the ultra-complex data-set produced by normal scheduling can be sufficiently reduced, sufficiently filtered, so as to be comprehensible by a driver.

Passengers, by comparison, are ‘consumers’ of the mobility services provided by drivers who, although they may well have some idea of where and when they will travelling on a given day, are to a significant

degree deciding at least the specific time of their movements on-the-fly. If passengers are making travel decisions on-the-fly then, if the ultra-complex data-set produced by ultra-efficient timetabling is to be sufficiently reduced so as to be comprehensible to them, then this data-set must also be reduced, must also be filtered, on-the-fly. The provision of an on-the-fly passenger information system, as part of deploying an ultra-efficient timetable, can therefore be seen as the equivalent of providing custom driver schedules, as part of deploying normal scheduling. Both custom driver schedules and on-the-fly passenger information systems provide tailored information to their respective information consumers in such a way that these consumers can effectively ignore the fact that, in totality, the information in question is ultra-complex.

The reason for using computationally intensive timetabling methods is the same as for using computationally intensive scheduling methods, namely that such methods produce more efficient outcomes. It is import to recognise, though, both that it would be passengers, the general public, ‘consuming’ the information produced by computationally intensive timetabling methods, and that they would be doing so in an on-the-fly manner. As such, very careful consideration would need to be given as to what information systems would be required to reduce that complexity, via on-the-fly filtering, to a simplicity level that could be sensibly deployed.

4.9 Conclusion

In conclusion, ultra-efficient timetabling has one key characteristic, the rapid tracking of passenger load, and two secondary characteristics, the use of computational optimisation and the barring of intra-line transfers. From the key characteristic of ultra-efficient timetabling, a transit system can be said to have been timetabled in an ultra-efficient manner if its services change so rapidly that passengers can no longer be reasonably expected to determine by themselves which service they need to board. This definition has been used to develop a method of ultra-efficient timetabling, the details of which will be the key subject matter of the next chapter. Having both a definition and a method for ultra-efficient timetabling allows the similarities to ultra-efficient scheduling to be seen, and strongly suggests that an ultra-efficient timetabling method will produce ultra-complex output. The consumers of the expected ultra-complex output of an ultra-efficient timetabling method are different though, and so, any deployment of an ultra-efficient timetable system would also almost certainly require the simultaneous deployment of a passenger information system of comparable sophistication.

< blank page >

< blank page >

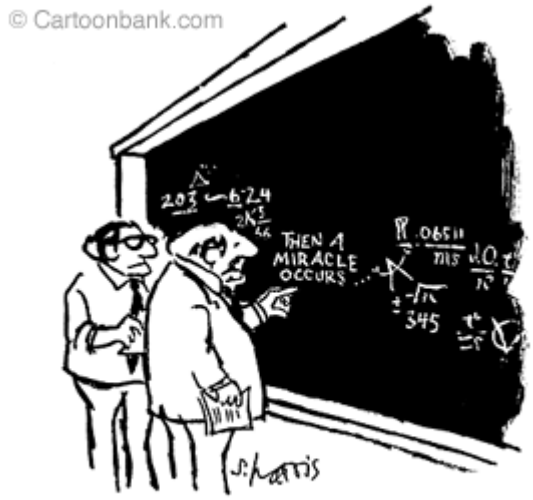
< blank page >

< blank page >

< blank page >

Chapter 5

Ultra-efficient Timetabling Software



"I think you should be more explicit here in step two."

156

5.1 Introduction

Programming is a specialist skill and as such it is often the case that programmers rarely attempt to explain how a complex piece of software they have written works, except to other programmers, and in language only other programmers can understand. Sometimes, when it seems particularly necessary to describe the operation of a piece of software only a general description is given, or only the design decisions embodied in the software are discussed, rather than the software itself. Sometimes even such talking around the subject is eschewed, and the operation of software is either discussed in terms of metaphors or, even when there is a clear need, not discussed at all.

156 *Sidney Harris : "I Think You Should be More Explicit Here in Step Two"*, Cartoon Bank Website, Available: http://www.cartoonbank.com/product_details.asp?sid=40967 (Accessed 23 May 2009)

It is a worrying thought that there might be a conspiracy of silence between those who write software and those who, although they don't need to write software themselves, really do need to understand how it works, beyond the level of metaphor or a vague general description. Both the task of clearly explaining the operation of software and the task of understanding those explanations are onerous ones. It is a concern that, in many cases, an unspoken agreement might be in place, with everyone agreeing that a detailed description of a piece of software's operation is not needed, when really it is. Consequently, where software is concerned, articles are often missing a section, meetings are missing an agenda item and larger documents a chapter; the one that actually says how the software works at the level of organising and transforming data.

With some care, describing the operation of much software, down to the level of actually organising and transforming data, to non-programmers, is entirely possible. Software is commonly built out of a surprising small number of basic keywords, also know as reserved words. For example, the programming language Pascal has the 35 keywords shown in Table 5.1 below.

and	array	begin	case	const
div	do	downto	else	end
file	for	function	goto	if
in	label	mod	nil	not
of	or	packed	procedure	program
record	repeat	set	then	to
type	until	var	while	with

Table 5.1: Pascal Keywords[157]

157 Jensen, K., Wirth N., Mickel, A. B. and Miner, J. F. (1991), P*ascal User Manual and Report (Fourth Edition)*, Springer-Verlag, New York, p. 10

Even such a list represents somewhat of a complexity overestimate, as a number of these keywords are rarely used, and the meaning of others, such as *if* and *while,* is exactly the same as the normal English language meaning. Similarly, there are a surprising small number of basic concepts, such as declaration, assignment and branching, used to build software, and concepts such as these are readably explainable in plain English. Correctly deploying such programming keywords and concepts may well take several years of specialist education followed by further years of practice; but being able to understand how a piece of software works does not imply the ability to write such software. It is certainly the case that the operation of the software developed for this thesis, though complex, is comprehensible to non-programmers. It has been decided that this thesis will not have have a 'missing chapter', that the operation of the software developed for this thesis will be described down to the level of organising and transforming data.

This chapter is primarily structured as a walk-through of the software written to generate an ultra-efficient BRT timetable for the TransMilenio's Américas Line. As such, it is intended that this chapter be read side-by-side with the code itself. This chapter will begin by outlining the coding approach that was used to develop the software. The code behind each of the software's menu options will then be walked through in turn, with, in the general case, one section being allocated to each menu item. The operation of the keywords out of which the software has been built, such as *if*, *for* and *while*, will be described as the terms are encountered. Similarly, a number of terms, such as span and link, which

have been used to describe certain logical aspects of BRT systems will be described as they are encountered.

5.2 Coding Approach

In developing software to timetable BRT systems in an ultra-efficient manner the first decision that needed to be made about the approach to be taken was how long the generation of those timetables would be allowed to take. When dealing with combinatorial problems, different approaches can mean the difference between a block of code that runs in minutes and one that will not complete in a lifetime. As was discussed in chapter two, this is a well understood circumstance, with problems that cannot be optimally solved in a reasonable amount of time being referred to as intractable. A decision was made early on that the maximum time allowed for a full one-day ultra-efficient timetable to be generated would be eight or nine hours on a single normal computer.[158] It was decided that the software was to be run on a single normal computer for the obvious reasons of practicality; normal computer time is far more readily available than super-computer time. The time limit of eight or nine hours, on such a normal computer, was chosen as this time period allowed a run to be generated once every day, overnight.

From this requirement to be able to generate a full timetable overnight came the name for the software developed for this thesis, “Matilda.” A “Matilda” in the Australian vernacular means “a roll of bedding and clothing which is carried across the shoulder by a traveller

158 Normal here refers to an average power consumer computer, as opposed to a workstation or super-computer. As detailed earlier, the specific virtualised computer used had a Intel Core 2 Duo 2 Ghz CPU with 384 megabytes of memory and was running Ubuntu Linux 9.0.4.

through the bush;"[159] or in more common parlance a bedroll. This name seemed apt for software that is required to complete its operation during the time that a person sleeps. For much of the time that Matilda was under development, full timetabling runs were indeed taking numerous hours and were mostly being run overnight. Towards the end of Matilda's development, optimisations were implemented that reduced that processing time down to approximately five minutes. The acquisition of a more powerful computer further reduced the processing time down to under one minute.

Matilda has been developed as a simple menu-based application, with letter coded options being presented to the user. Matilda's menu is shown in Figure 5.1 below.

```
matilda - ultra-efficient brt timetabling

menu options ( make a choice and hit return )

a - calculate line ratio
b - convert transmilenio run files to matilda run files
c - convert transmilenio load files to matilda load files
d - make run groups from loads
e - deadhead runs
f - make pax groups from run groups
g - make pax groups from loads
h - produce pax group analysis data
i - produce run group analysis data
q - quit

menu choice:
```

Figure 5.1: Matilda's Menu

On selecting a menu item any data that is needed to perform the requested data manipulation is read-in from human-readable text file(s), and after the requested data manipulation is complete any output data generated is written-out to human-readable text file(s). Not maintaining data internally means that data is sometimes written-out only, on the

159 *Botanic Gardens Trust - The Coolibah song - Waltzing Matilda*, Botanic Gardens Trust Website, Available: http://www.rbgsyd.nsw.gov.au/welcome_to_bgt/quick_links/kids_zone/stories_and_songs/the_coolabah_song__waltzing_matilda?SQ_DESIGN_NAME=printer_friendly (Accessed 23 May 2009)

selection of a further menu item, to be read straight back in again; an inefficiency. By always writing out the data, though, Matilda can be quit at any time, without have to manually ‘save’ data beforehand, and furthermore the current state of any sequence of calculations is available to be viewed in the human-readable text files at any stage of the calculation sequence. In computing, this is known as using a stateless approach, which “means there is no record of previous interactions and each interaction request has to be handled based entirely on information that comes with it.”[160] The selection of any of the above menu items causes one or more functions[161] to be called[162] to perform the work described by the menu item. When the work is complete Matilda re-presents the menu.

5.3 Calculating the Line Time

To develop a timetable for a set of buses servicing the Américas Line it is necessary to know how those buses would perform on the Américas

160 *What is Stateless?*, Tech-target Website, Available: http://whatis.techtarget.com/definition/0,,sid9_gci213051,00.html# (Accessed 17 January 2009)

161 A function is a named block of code that receives certain data as inputs, performs certain manipulations on that data, and may also return certain data as outputs. An example would be a function called MultiplyTwoNumbers, which receives two numbers as data input, performs a multiplication operation on those numbers and returns the resulting number as a data output. Normally functions perform more complex operations than a simple multiplication, often far more complex operations, but the concept of a function as a named block of code that performs a given task, and that has given inputs and outputs, remains constant. The approach used to name the MultiplyTwoNumbers example function given here, using capital letters to delineate the three words rather than spaces, is one of the normal ways of naming software functions, and is used throughout the Matilda software. Spaces are not used as part of the function name because a space character would be interpreted as the end of the function name, and the beginning of a new component of the software code.

162 To call a function is to direct that function to perform its computations and, in effect, to wait until the function has finished before continuing.

Line if they were to execute the timetable. Some examples of performance characteristics that are necessary to know include the operational speed of the buses, bus acceleration and deceleration capability and passenger board and alight time. The complete list of performance characteristic needed to develop a timetable can be worked out by breaking down a bus trip between two stops into its component parts as shown in Figure 5.2 below. The method shown below draws significantly on material from the *Travel Time Equations and Diagrams* section of Professor Vuchic's *Urban Transit Systems and Technology* book.[163]

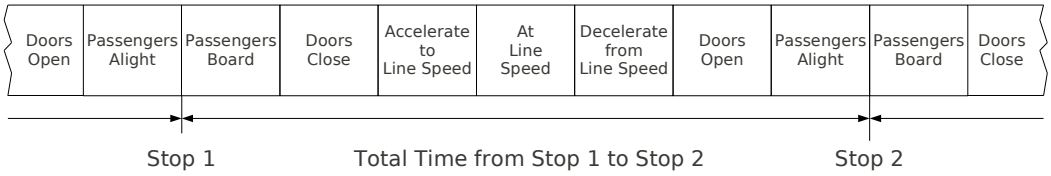


Figure 5.2: Parts Between Two Stops

Figure 5.2 shows the sequence of events between two adjacent stops, stop 1 and stop 2. There is no significant difference, though, between this situation, where the stops are adjacent, and the sequence of events between non-adjacent stops; if stops are being bypassed by express buses then then the time spent "at line speed" is simply greater. All of the data needed to work-out the times from each component of the above sequence of events was either obtainable from the literature or was obtained by taking measurements on the TransMilenio systems itself, during a site visit.

According to the *Bus Rapid Transit Planning Guide* the boarding times for the TransMilenio system is in the area of 0.3 seconds per

163 Vuchic, V. R. (2007), *Urban Transit Systems and Technology*, John Wiley & Sons, New Jersey, pp. 124-130

passenger.[164] This figure was confirmed by on-site observation, and also the alighting times were observed to be similar to the boarding times. This comparatively fast boarding and alighting is due to the pre-boarding ticketing, level-boarding and the “four sets of 1.1 metre wide doors” used on the TransMilenio system.[165] On the TransMilenio system the boarding and alighting of passengers is not as ‘tidy’ as the neat separation into boarding and alighting times shown in Figure 5.2 suggests, with boarding and alighting often taking place at the same time. For the purposes of timetabling, though, the key issue is to account for the boarding/alighting time taken by each of the projected passengers, so the simplification shown in Figure 5.2, of boarding and alighting happening as discreet events, is workable. With regards to the timetabling in this thesis each passenger boarding event and each passenger alighting event is set as taking 0.3 seconds.

Door opening and closing time was assessed, on site, by taking the average of ten time measurements. Door opening time was measured as the time between when the bus had fully stopped and the time when the doors were fully open. Door closing time was measured as the time between when the doors began to close and the time when the bus began to move off. Door closing time came out, on average, as 4.9 seconds and door opening time came out, on average, as 3.5 seconds.

The time taken to accelerate to line speed and decelerate down from line speed was, in the same manner as the door opening and closing time, measured on site, by taking the average of ten time measurements. The

164 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York. p. 262

165 ibid.

time taken to accelerate up to line speed was measured as the time between when the bus first began to move off and the time when the bus reached line speed. The time taken to decelerate down from line speed was measured as the time between when the bus first began braking down from line speed and when the bus was fully stopped. The time taken to accelerate up to line speed came out, on average, to 31.6 seconds and the time taken to decelerate down from line speed came out, on average, to 13.8 seconds.

Line speed itself was assessed on site as being 60 km/h, a level that was confirmed during a meeting with Alejandro Niño of the TransMilenio S.A. Planning Department. From this meeting, the reason for this speed limit is that it is the general speed limit for roadways in Bogotá and so therefore no special legal consideration was required to operate up to this speed. It was observed on site that the TransMilenio drivers stuck to this 60 km/h speed limit with great precision, almost to the point of not one km/h slower or faster.

The line speed level completes the set of numbers that are required to allocate a magnitude level to the components of total bus travel time that were presented in Figure 5.2. In Figure 5.3 below, these components and numbers are shown together.

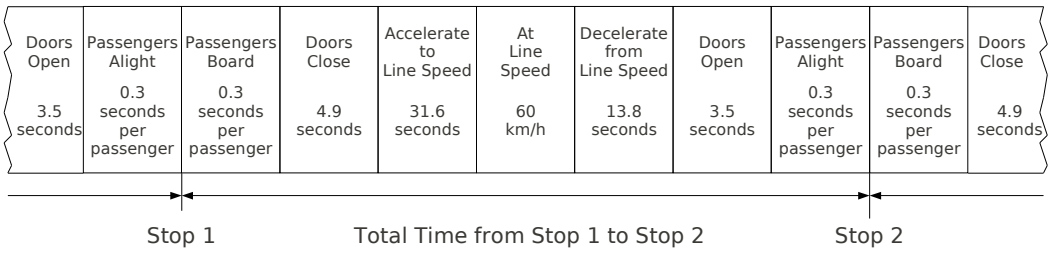


Figure 5.3: Parts Between Two Stops (with Magnitudes)

All of the above magnitudes for the components of total bus travel time are in time units except the at line speed magnitude, which is given in km/h. To convert this km/h magnitude into time units it is necessary to know the distance that is being spent at line speed. This distance is, by definition, the total distance between two stops minus the distance taken to get up to line speed minus the distance taken to get down from line speed. In Figure 5.3 above the time taken to get up and down from line speed is given as a duration of time, not as a distance. To convert time into distance it is necessary to know the acceleration and deceleration profile. In line with the U.S. Federal Transit Administration's *Characteristics of Urban Transportation Systems* Report[166] I have assumed deceleration to be constant, which gives, for the line speed of 60 km/h over the deceleration time of 13.8 seconds, a deceleration distance of 115 metres. The same U.S. Federal Transit Administration's *Characteristics of Urban Transportation Systems* Report[167] also gives a typical acceleration profile for buses of 5.36 km/h^2 from 0 to 16 km/h, 3.57 km/h^2 for 16 to 48 km/h and 1.53 km/h^2 for 48 to 80 km/h. Using these numbers gives an acceleration distance of 204 metres.

If the TransMilenio's Américas Line were an idealised transit line, with no line condition issues to complicate the situation, then the above data would be sufficient to model the line's behaviour. The Américas Line is, though, a real transit line that has been retrofitted to an existing urban environment, and consequently there are two line condition issues; one minor and one major. The minor issue, speed restrictions due to tight

166 U.S. Federal Transit Administration's Characteristics of Urban Transportation Systems Report, Chapter 3: Bus Transit

167 ibid.

line curvature, is common to both BRT and rail based transit systems. On the Américas Line there is a single speed restriction, down to 20 km/h, between Pradera and Marsella stops. The major line condition issue on the Américas Line, and the TransMilenio system as a whole, is the substantial number of traffic lights. On the 13.3 kilometres of the Américas Line there are 16 car-intersection traffic lights and 8 pedestrian-intersection traffic lights. This large number of traffic lights has a significant impact on transit speed therefore moves the model for the Américas Line away from the idealised one presented in Figures 5.2 and 5.3. Consequently, to be able to model the Américas Line in a realistic way, one that allows realistic timetables to be constructed, the impact of the line condition issues on the Américas Line need to be quantified.

The approach that has been taken to incorporating line condition issues into the model has been to calculate the time difference between an idealised line and the real one and to add that time difference, as an adjustment level. Two methods were available for calculating an adjustment level. As this is an important issue, both were calculated.

The first method used was to isolate buses in the received TransMilenio data that were being moved around without passengers, for the purposes of starting and stopping services, and to compare their transit times to the idealised transit time. When a number of buses were being moved from a depot to the outermost stop, the average time taken for this trip was 5:46 minutes. Similarly, when a number of buses were being moved from the same depot to the innermost stop the average time taken for this trip being 30:00 minutes. Subtracting one time from the

other gives a reading on time taken to travel the whole line, with the whole line being by definition from the outermost to innermost stop, of 24:14 minutes.

An idealised trip along the length of the Américas Line would involve accelerating up to line speed, travelling until the point was reached to start decelerating down to a stop, and then decelerating down from line speed. The full length of the Américas Line is 13,296 metres. As previously presented, the distance taken to accelerate up to line speed is 204 metres and the distance taken to decelerate down from line speed is 115 metres. Subtracting these two figures from the total line length gives a distance at line speed level of 12,977 metre, which at 60 km/h would take 12:59 minutes. Adding the time taken to get up to line speed of 31.6 seconds and the time take to get down from line speed of 13.8 seconds, gives a total idealised line travel time of 13:44 minutes. Subtracting this idealised line travel time from the actual line travel time of 24:14 minutes gives a line condition time level of 10:30.

An alternative method for calculating line condition time is to compare not an empty bus run to its idealised counterpart, but to a service run, one that transports passengers to its idealised counterpart. Express 120 is a good candidate for doing such a comparison as it runs the whole length of the line and for almost the whole of the day. All of the data required to do such a comparison is at hand except the expected number of boardings. The closest available stand-in for a boarding number is occupancy, which can be estimated using the average line occupancy. From the data there were 1,409,000 passenger kilometres and 4,753,280 place kilometres on the Américas Line, giving an average

occupancy of 29.6%. To convert this occupancy level into a boarding level estimate it is necessary to know the expected ratio between occupancy and boarding for a given stopping pattern, in this case the Express 120 stopping pattern. Calculating this ratio required a short function, called CalculateLineRatio, to be written.

Menu option “a” of the Matilda software, “calculate line ratio”, which can been seen at the top left of the Figure 4.7 data-flow diagram, calls the CalculateLineRatio function. The CalculateLineRatio function starts by reading in the average inter-stop passengers loads for the Monday to Friday period. The passenger loads on each of the inter-stop links[168] serviced by Express 120 are then accumulated by running over all time periods. The total load on Express 120’s inter-stop links is then calculated, and the load on each inter-stop link divided by this total, thus reducing the total load down to, in effect, one passenger. From the fractional loads on each inter-stop link, the distance this notional passenger travels can be calculated by multiplying the fractional load by the inter-stop distance. This distance is outputted as a percentage of the total line length and came out to 71.74%.

168 The term “link” is used throughout the remainder of this thesis and throughout the Matilda code to refer to a trip between two given stops. So, for example, the trip between Portal de las Américas, the outermost stop on the line, and Avenue Jiménez, the innermost stop on the line would be one link. Links are unidirectional, meaning that the trip between Portal de las Américas and Avenue Jiménez, and the reverse trip between Avenue Jiménez and Portal de las Américas are two different links. Notionally the total number of links is the square of the number of stops, so with there being 18 stops on the Américas Line there are notionally 324 links. Of these 324 links, though, 36 are between the same stop and so cannot be travelled in any meaningful sense, leaving 288 links able to be travelled. In numerous places throughout the Matilda code, passenger load is organised by reference to the link the load is occurring on, rather than by the two stops the load is occurring between.

If, on average, when one passenger boards they travel 71.74% of the distance of the line, it can be calculated that if 1.394 passengers board they will travel exactly the length of the line. An occupancy of 29.6% and a bus capacity of 160 means that 47.42 are, on average, travelling the length of the line. With a boarding level of 1.394 per full line travel, this means the average number of boardings for an Express 120 bus can be estimated as being 66.11 boardings. With this number the idealised travel time for Express 120 can be calculated.

Express 120 stops at four intermediate stops and so it needs to accelerate up to and down from line speed five times. The distance taken to accelerate up to line speed is 204 metres, with five times that figure being 1,020 metres, and the distance taken to decelerate down to line speed is 115 metres, with five times that figure being 575 metres. With the Américas Line being 13,296 metres long, this means that an idealised Express 120 will be at line speed for 11,701 metres. At the line speed of 60 km/h, 11,701 metres would be travelled in 11:42 minutes. Add five times the 31.6 seconds required to get up to line speed and five times the 13.8 seconds required to get down from line speed and you have a total travel time of 15:29 minutes.

Express 120 buses, though, are not just travelling from one end of the line to the other, they are also transporting passengers, which requires them to open and close their doors and to allow sufficient time for passengers to board and alight. As Express 120 buses stop at four intermediate stops they need to open and close their doors five times. Five times the door opening time of 3.5 seconds is 17.5 seconds and five times the door closing time of 4.9 seconds is 24.5 seconds; add these

amount to the travel time of 15:29 minutes gives 16:11 minutes. This only leaves the passenger board and alight time to be accounted for. As previously calculated an average of 66.11 passengers will board and alight Express 120 buses. At 0.6 seconds per passenger this gives a total board and alight time of 40 seconds which if added to the current run time of 16:11 minutes gives a grand total idealised run time of 16:51 minutes for Express 120 buses. From the data the actual average run time is 27:00 minutes, which gives a line condition time level of 10:09.

The two different methods have been used to calculate a figure for the Américas Line's line condition time have yielded reasonably similar figures of 10:30 and 10:09 minutes. A 21 second time difference over approximately a 24 to 27 minute time span is not great, but for the sake of taking the more conservative approach the higher, and therefore slower, 10:30 minutes figure has been used for the ultra-efficient timetable development. In terms of implementation, this 10:30 line condition figure has been utilised by 'spreading' it out evenly across the line in 17 equal quantities of approximately 37 seconds, one between each of the 17 inter-stop spans.[169] In terms of execution, to calculate the travel time between two stops the idealised travel time is calculated and then 37 seconds is added to that figure for each of spans between the two stops.

169 The term "span" is used throughout the remainder of this thesis and throughout the Matilda code to refer to the zone between two adjacent stops. Spans are numbered using the lower of the two stops they are between. So the span between stop 0 and 1 would be span 0. Spans are non-directional so the span between stop 0 and 1 is the same as the span between stop 1 and 0. Spans are used to record data such as the distance between two stops. As spans occur between adjacent stops, there are 17 spans on the 18 stop Américas Line.

5.4 Converting the TransMilenio Run Files

In section 3.3 of this thesis, “The ‘Raw’ Bus Runs Data”, the nature of the bus run data received from TransMilenio S.A. was discussed and in section 3.4, “Isolating and Constructing the Bus Runs”, the data manipulation to be performed on this data was covered. This section discusses the technical detail of how the data received from TransMilenio S.A. was processed and the outputs generated by that processing. As discussed in section 3.4, the bus run data to be used by this thesis is a single file that contains every timetabled bus movement event for the timetable being run across the Monday to Friday period. The Américas Line data is isolated from the system as a whole by truncating the two express services that run beyond the Américas Line, namely express service 80 and 100, to Avenida Jiménez, the innermost stop on the Américas Line.

Menu option “b” of the Matilda software, “convert Transmilenio run files to Matilda run files”, which can been seen at the centre left of the Figure 4.7 data-flow diagram, calls a single function, namely ConvertRunData. Along with the previously mentioned task of isolating the Américas Line from the rest of the TransMilenio system, the function ConvertRunData is also responsible for converting the received movement-event files into run files. The data from TransMilenio S.A. records bus movement events, in other words events that happen to buses at a particular moment in time, such as a bus leaving a stop at a given time. The run analysis system of the Matilda software has been written to process bus runs, not events, thus the need for the conversion. Bus runs happen over a time periods. For example, an express 120 bus

starting its run at 9:30 am and finishing that run at 9:57 am, would be a single run but at minimum two events, the 9:30 am start and 9:57 am finish.

The data that the ConvertRunData operates on are four files, in the same format as the one file received from TransMilenio S.A.; one file for each of the four services, one all-stop and three express, that run on the Américas Line. These four files were split off from the single received file using basic spreadsheet filtering techniques, with the filter being applied, in this case, to the “Line” column which gives a service number for each of the 16 services on the TransMilenio system as a whole. Each of these four files was sorted first by bus index number, smallest to largest, and then by time, earliest to latest.

For each file, a one dimensional array[170] called bus_events[171] is declared,[172] with the dimension being the bus index number and the array

170 The most commonly used type of array, outside of programming, is a spreadsheet. A spreadsheet is a two dimensional array, having an x and y dimension and a grid of cells which contain data for each x and y dimension intersection point. Normally the x and y dimensions refer to something. For example, the x dimension might be Student IDs and the y dimension might be Study Unit IDs, allowing a data element to be stored for each combination of Student ID and Study Unit ID. For example, one might store the year that each student completed each study unit in the data cell, leaving the cell blank if the unit has not been completed. An array in programming is exactly the same as a spreadsheet, except that it can have either less or more dimensions than the two used by spreadsheets. In the example here the array only has one dimension, making it a list.

171 An underscore character, rather than a space, is used between the words “bus” and “event” here, for the same reason that space characters are not used in naming function, namely that a space character would be interpreted as the end of the name, and the beginning of a new component of the software code.

172 In programming to declare something is to specify its characteristics and to give it a name. In this example, the characteristics of the thing being specified here include the fact that it is one dimensional array, and the name being given to this one dimensional array is “bus_events.”

cells being declared to each be a list[173] of strings.[174] The array is filled by each text line of each bus movement file being read in, the bus index isolated, and the text line placed into the list of strings, at the point in the array specified by the bus index. As the data stored in the incoming bus movement files has been pre-sorted the list of strings is also sorted, with the earliest movement for each bus being at the top of the list and the latest movement at the bottom of the list. So, for each file and for each bus in that file, a list is constructed of all of that bus's movements, with the list of bus movements being sorted from the earliest to latest movement.

A bus run is delineated by two consecutive bus events, a departure event from an origin stop and an arrival event at a destination stop. The process of converting bus events into bus runs is therefore begun by pairing each bus event with the bus event that follows it. To achieve this pairing a one dimensional array called bus_runs is established, with the dimension again being the bus index number and the cells again each being a list of strings. For each of the buses, each of its movement event strings is paired with the next movement event strings, with the strings separated by a tab, and the resulting line is added to the bus_runs array, at the point in the array specified by the bus index.

The above processing pairs bus events to generate bus runs, but not all paired events are runs. For a bus to leave a stop it first has to arrive, and some of the pairs specify a bus arriving at a given stop and then

173 The term "list" is being used here, and throughout the remainder of this chapter, as a appropriate common English description for what in the programming language used for this thesis is technically called a vector. In the code, the actual term vector is used. A vector is the name for a type of list that does not have any automatic sorting applied to it.

174 In programming "string" is short for string-of-text-characters. A string is a piece of text, so a list of strings, is a list of pieces of text.

leaving from the same stop. In the case where there is no delay timetabled between a bus arriving at a stop and leaving the TransMilenio data only records the leaving event, but in the common case where a delay, usually of around a minute or so, has been timetabled both the arriving and leaving events are recorded. As the MakeRunGroups function is only interested in runs, not stop-based time periods, event pairs that occur at the same stop are discarded.

Also discarded are any trips between depots and either the outermost stops, namely stop 17 for both the all-stop service and expresses 100 and 120, and stop 13 for express 80. The timetable development in this thesis treats stop 17, on the line, as the depot stop, whereas the existing May 2005 efficient timetable pushes and pulls buses from depots beyond the lines of the TransMilenio system. To match the May 2005 timetable data as closely as possible to the timetable constructed in this thesis, all travel from and to depots is either discarded, as here, or trimmed back to the line. To do otherwise would leave the existing efficient timetable with the task of moving buses between the line and the depots, whereas the ultra-efficient timetable constructed in this thesis would not have that task, thus skewing any comparison between the two approaches in favour of the ultra-efficient timetable.

In cycling through each bus, and then each event pair for each bus, if either of the above two cases where event pairs need to be discarded are met, the code that converts event pairs into runs is not executed. For all event pairs that are runs, the event pair conversion code is executed. Not all of these runs are bound to the Américas Line, two of the express services, express services 80 and 100, run beyond the and so need their

runs to be trimmed back to the Avenida Jiménez stop, the last stop on the Américas Line. Express service 120, the single express service on the Américas Line that does not travel beyond the line, was used as guide to the general speed of express services on the Américas Line, with express services 80 and 100 being assumed to travel at the same speed along the Américas Line as express service 120. As was discussed in section 3.4 “Isolating and Constructing the Bus Runs”, the three express services stop at very similar stopping intervals along the Américas Line, and so assuming the same travel speed for the express services is not unreasonable.

Express services 80 and 100 are trimmed back to the Américas Line by isolating the case where a service starts on the Américas Line and ends somewhere off the Américas Line, and the reverse case. Once these two cases have been isolated the express service, 80 or 100, is identified via the file number that is currently being processed. Once express run cases that need to be trimmed have be identified, either the from (departure) or to (arrival) time is adjusted to be what it would be if these services were stopping or starting from the Avenida Jiménez stop, the last stop on the Américas Line. The adjustment time for inbound and outbound services is slightly different, reflecting a slight difference in the speed data for express service 120, the express service being used as a guide to the speed of the other two express services on the Américas Line. For deadhead runs, runs to and from depots, the deadhead speed of express service 120, rather than the passenger service speed, is used to trim the runs of express services 80 and 120.

The final task performed by the ConvertRunData function is to calculate the bus drain levels, the number of buses that are in use at any one time. This task is not strictly data conversion, but due to the data already being manipulated by the ConvertRunData function, performing these calculations here was the most straight-forward approach. To calculate the bus drain level a list of depot departure and return times, called run_counters, was constructed. A magnitude of plus one was associated with a departure event and a magnitude of negative one associated with a return event. This list was then sorted from the earliest time to the latest time, and a counter variable called run_level declared and zeroed. By running through the depot departure and return events listed in the variable run_counters, and adding or subtracting one as buses respectively departed or returned to depots, the total number of buses on the line at any given time could be known. The list of changes in bus levels is both displayed to the user and written out to a file. This completes the work of the ConvertRunData function.

5.5 Converting the TransMilenio Load Files

In section 3.6 of this thesis, “The ‘Raw’ Passenger Load Data”, the nature of the load data received from TransMilenio S.A. was discussed and in section 3.7, “Isolating and Constructing the Passenger Load”, the data manipulation to be performed on this data was covered. This section will discuss the technical detail of how the data received from TransMilenio S.A. was processed and the outputs generated by that processing. As discussed in section 3.6 the primary load data to be used by this thesis are five passenger movement tables which record, for every

TransMilenio stop Monday to Friday, the number of passengers who enter and leave the stop every 15 minutes. The Américas Line data is to be isolated from the system as a whole using some secondary origin-to-destination data, received from TransMilenio S.A., which allows the likelihood of travel between the Américas Line and the rest of the TransMilenio system to be calculated.

Menu option “c” of the Matilda software, “convert Transilenio load files to Matilda load files”, which can been seen at the top centre of the Figure 4.7 data-flow diagram, calls two functions, namely ConvertLoadData and MakeLoads. The function ConvertLoadData is responsible for isolating the Américas Line from the rest of the TransMilenio system and for converting the received stop-event files into origin-to-destination files. The function MakeLoads is responsible for generating a single average origin-to-destination load file from the Monday to Friday files, and for scaling that file up to the line load level of 186,000 boardings.

The function ConvertLoadData performs the same processing procedure for each day, Monday to Friday. Before each of the TransMilenio S.A. data files are read in, a three dimensional array is declared, with the three dimensions being the time slice[175] index, the stop

175 As was previously discussed in this thesis, much of the received data was in 15-minute time increments and the production of the ultra-efficient timetable will be done in 15-minute time increments. These 15-minutes increments are called time slices in the code. With there being four 15-minute time slices in one hour and 24 hours per day there are 96 time slices per full day. Each of these 96 time slices has been allocated an index number, using the coding practice of stating indexes from 0 rather than 1, the time slice indexes run from 0 to 95. The approximate midpoint of the TransMilenio line close time period, 3:00am, was chosen as the start point of the time slice index; so time slice index 0 runs from 3:00am to 3:15am, time slice index 1 runs from 3:15am to 3:30am and so on.

index and the entry/exit status. The floating-point[176] value stored in each cell of this array is the passenger load level for that cell, which for all cells in the array is initialised[177] to 0. As each TransMilenio S.A. day-load file is read in, it is converted into separate lines, with each line being one stop entry or exit load level for one stop during one time period. The line is broken into its component elements and the file line elements are converted into three indexes, one for each of the three dimensions of the array, and a passenger load level. As part of isolating the Américas Line from the rest of the TransMilenio system, any stop that is external to the Américas Line is converted to being stop 0, the innermost stop, which thereby has the innermost stop representing the entire TransMilenio system beyond the 17 outermost Américas Line stops, which are Américas Line only stops. To complete the isolation of the Américas Line from the TransMilenio system, the passenger load on stop 0 is then scaled down in accordance with the travel between the Américas Line and the rest of the TransMilenio system. Finally, the passenger load level is added to the array, using the three indexes extracted from each file line to select the cell in the array to which to add the load.

This process of shifting file data into an array gets the data in a ready manipulable form, and in doing so separates the Américas Line from the rest of the TransMilenio system, but the data is still stop entry and exit load data and it is inter-stop load data that is required. The conversion process is begun, as with reading in of the file data, by establishing a three dimensional array, with the three dimensions being the time slice

176 A floating point number is a number that is not necessarily an integer.

177 To initialise an array cell is to set its initial or default value. If an array cell is not initialised it will contain whatever random value happen to be in the computers memory.

index, the from-stop index and the to-stop index. Each time slice is treated one at a time and for each stop during a given time slice the total stop exit load is added up. The inter-stop load is then calculated on the basis that the load on a given inter-stop link is the from-stop entry load level multiplied by the ratio created by dividing the stop exit load by the total stop exit load. In other words, for a given time slice if 10% of people are exiting from-stop B, then when a person enters any other stop it will be assumed that there is a 10% chance they will exit from-stop B. The case of entering and exiting the same stop, has been dealt with by setting the load level to zero. Once all time slices have been processed, and all inter-stop load levels in the array have been set, the last task performed by the function ConvertLoadData is to write the array out to a file; as this process is happening for five days, Monday to Friday, five files are created.

Once the ConvertLoadData function completes the MakeLoads function is called. The MakeLoads function starts by establishing a three dimensional array of the same structure as the now processed Monday to Friday load data, specifically the three dimensions being the time slice index, the from-stop index and the to-stop index. This array is established to store average load data and is zeroed before use. Each Monday-to-Friday load file is then read in and its load values are added to the array established to contain average load data. The total load of this average load data array is then calculated and a scaling ratio is calculated by dividing the line load, of 186,000 boardings, by this total load figure. The scaling factor is applied to all of the data in the average load data array, to bring the total load down to 186,000 boardings, and then the average

load data array is written out to a file. This completes the conversion of the five TransMilenio S.A. day stop-event based load files into a single average-load inter-stop passenger load file.

5.6 Making the Runs: Data Preparation and Pre-calculations

Menu option “d” of the Matilda software, “make run groups from loads”, which can been seen near the centre of the Figure 4.7 data-flow diagram, calls the function MakeRunGroups, which performs the central task of the software, namely tailoring an ultra-efficient set of timetables for a Bus Rapid Transit line. All of the other functions in the Matilda software, with the exception of the run deadheading function, are support functions to the MakeRunGroups function. The functions before the MakeRunGroups function are setting up the data for this function to operate on and, with the exception of the run deadheading function, the functions after the MakeRunGroups function are, in some manner, preparing the data generated by that function for analysis. The run deadheading function is an exception to the other functions surrounding the MakeRunGroups function because the MakeRunGroups function does not attempt to ‘balance’ bus flows, and as such the run deadheading function can be seen as phase two of the run making processes.

Due to the complexity of the MakeRunGroups function its operation will be discussed over three sections, this sections (5.6) and two subsequent sections 5.7 and 5.8. This section will discuss how the data, collated and constructed as per the methods outlined in the proceeding section, was prepared for use to construct runs, and the pre-calculations that were necessary before the task of constructing runs could be

conducted. After this section, section 5.7, “Making the Runs: Making Run Groups”, will discuss how, for a given time slice and direction, a particular set of express runs are selected from the vast array of options. Finally, section 5.8, “Making the Runs: Making a Single Express Run”, will discuss how a single express run is selected, a task that has been in large part isolated into a separate function called make_one_run_group.

The function MakeRunGroups begins by pre-calculating some data which is needed when using some of the functions called by the MakeRunGroups function. Firstly, the spans for each of the Américas Line’s links are calculated, and stored in an array called links_spans. Links are between any two stops whereas spans are only between adjacent stops, and so for example, the outbound link between stops 5 and 7, would consists of the outbound span 5, between stops 5 and 6, and the outbound span 6, between stops 6 to 7. Next, all of the load carrying links, those not between the same stop, for the Américas Line are calculated, and stored in an array called links_far_all_stops_pos.[178] Finally, all of the stops for the Américas Line are calculated, and stored in the variable called stops_far_all_stops.

To begin the process of calculating the runs, a list for the runs, called runs, is declared and the load data, constructed as described in the previous section, is read from the file into a list called from_to_loads. As

178 Whenever the words “near” or “far” appear in the code, as the word “far” has is the array name here, what is being referred to is the stop that buses are turning around at, whether it is the near stop, stop 13, or the far stop stop 17. The word “pos” in the array name here is short for positive, as opposed to negative which also appears in the code as “neg.” When the term positive is used in the code it refers to the links that are demarcated by a given stopping pattern. When the term negative is used it refers to the links that are not demarcated a given stopping pattern; those links left unserviced by a given stopping pattern. The all-stops case here covers all links, and so this list of positive (serviced) links would contain the complete list of links. Consequently, if a list of negative (unserviced) links were to be constructed for this all-stop case, it would be an empty list.

previously discussed, the time slice for the load data was set from the entry-stop load data, with the exit-stop load data only being used to break up the entry-stop loads into their respective exit-stops so that origin-to-destination load levels could be established. As such, the time datum for the load data, at this point, is the entry stop. In other words, a load level for a given time period specifies a load that is occurring at the entry stop, not a load that it is occurring either at the exit stop, or at some notional point on the transit line.

For load to be measured it must be measured at some point, but in the context of calculating bus runs, measuring the load at the entry stop is less than ideal. For the purposes of calculating bus runs, the central concept in play is load along the line, not load at the entry stop, or for that matter the exit stop. Within time constraints, bus runs try to maximise the use of their seats along the line, and stopping to let passengers on and off is just a way of achieving seat utilisation. As will be discussed in more detail later, the centre-point of the line was chosen as the best datum point from which to calculate bus runs, and consequently, in this context, the centre-point of the line is the best time datum for the passenger load data.

To convert load data from having an entry stop time datum to having a line centre-point time datum, requires the travel time between each stop and the centre-point of the line to be known. For example, if between 06:00 and 06:15 at a given stop there is a given inter-stop load of six passengers, then if the travel time from that stop to the centre of the line is 15 minutes, it can be said that at the centre of the line that inter-stop load of six passengers occurs between 06:15 and 06:30, fifteen

minutes later than at the stop. This in principle simple concept, is made complicated by the fact that travel time to the centre of the line varies; different express stopping patterns and different load levels will give different bus travel times. For the purposes of moving the load data time datum to the centre of the line, an average line travel time was used, and as before the average line travel time was set at thirty minutes. As deviation from this average will on average move load around by a matter of only a few minutes, without adding or removing load, this necessary approximation was considered acceptable.

The actual conversion of the load data from having an entry-stop time datum to having a line centre-point time datum, is done by a function called OffsetLoadsToLineCentre, run on the load data immediately after the load data is read in. Given the relative simplicity of the OffsetLoadsToLineCentre function, and the extensive discussion of the required task given above, a detailed break-down of this function's operation is superfluous. In brief, the OffsetLoadsToLineCentre function declares a new empty load array, cycles through each inter-stop pair for all of the time slices, adjusts the inter-stop load by the required time-offset and then copies the time adjusted load level into the new load array. Finally, by over-writing the original load level with the new time levels, the adjustment of the load data time datum to the centre of the line is completed.

Moving the time datum for load data is the major task that is conducted to convert the load data from a general data source into a form better suited to the task of calculating ultra-efficient bus runs. Subsequent to this task a minor task is also done, namely moving the

load data from being accessed via its from and to stops, to being accessed via its link IDs, and restoring the load data in a list called link_loads. This change is done purely because it is easier, in this context, to work with link IDs and, unlike the previous manipulations does not change the load data. Before the link loads data is used, to ensure the correct load of 186,000 passengers is used, the total load is calculated and printed out. This completes the process of preparing the passenger load data.

The next task is to calculate the number of runs and channels for both the inbound and outbound direction for every time slice. As per the standard nomenclature, the use of the term “run” here refers to the buses that are dispatched to service passenger load. The use of the term “channel” here refers refers to the number of service patterns that are being deployed. For example, if there was one all-stop service pattern and two express patterns deployed during a given time period then there would be three channels operating during that time period. As has been previously discussed, the time division being used for this timetable development exercise is 15-minutes, so all run and channel levels are the run and channel levels for a given 15-minute time period.

Setting the channels and runs does not in itself set how these runs are to be distributed amongst the channels. If, for example, there were 14 runs and three channels, one all-stop and two express, then three runs could be allocated to the all-stop channel, four to the first express channel and seven to the second all-stop channel, or these runs could be allocated in a myriad of other ways. The allocation of runs to channels is done by later algorithms, and is intimately tied in with the optimisation

process. The details of how runs are allocated to channels, and the constraints under which that allocation is performed will be discussed in section 5.7 “Making the Runs for One Direction and Time Period Only.”

The process of calculating the number of runs and channels is begun by establishing two three-dimensional arrays, called num_runs and num_channels, with the “num” being short-form for “number of.” The three dimensions of both arrays are the direction, inbound or outbound, all time slices and the terminating stop location, either the near or far turn-around stop. Stored in the cells of both of the two arrays is a floating-point number, that records the number of runs for the num_runs array and the number of channels for the num_channels array.

The number of channels and runs for each time slice and both directions are calculated independently, with all time slices and the two directions for each time slice being traversed. For each time-slice/direction pair, if there is no passenger load then both the number of runs and number of channels are set to zero. Alternatively if there is some passenger load then the process of calculating the number of runs and number of channels for near and far stop options is begun by calculating the total number of runs and total number of channels for the near and far stop options combined.

The total number of runs is calculated by dividing the peak passenger load by the capacity of the buses, namely 160 passengers, and rounding the resulting level up. If the resulting level is less than the minimum run frequency level then the total number of runs is set to be the minimum frequency level, so as to guarantee a minimum level of service regardless of whether there is the load to justify the run frequency. The minimum

run frequency level is determined by the maximum amount of time which it is acceptable for passengers to wait. In normal timetabling practice, with limited use of express buses, a minimum level of service is important to avoid passengers having to wait unacceptable periods of time. In the heavy-load BRT case, where substantial use of express buses is made, it could be argued that observing a minimum level of service is even more important, as the temptation to extend passenger wait time is greater. When choosing from a range of express bus options, a highly efficient option may be available so long as sufficient time is allowed to elapse that enough passengers are available to fill the bus. If this notional "sufficient time" is say two minutes as opposed to another option at one minute, then the two minute option may well be fine. However, it may well be the case that highly-efficient solutions exist if, say, eight hours are allowed to elapse before boarding the passengers. Eight hours is obviously too long, but eight minutes might not be; there is no 'right' answer here, only subtle questions of policy and practice. The Metropolitan Council of Saint Paul, Minnesota has, for example, in their *Metropolitan Council 2030 Transportation Policy Plan*, set different minimum service frequencies for both different services and different areas.[179]

When determining such questions of policy and practice this thesis uses, where-ever possible, the real-world May 2005 TransMilenio timetable as a guide. This approach assists in keeping what is a highly theoretical timetable development exercise as close as possible to the

179 *Metropolitan Council 2030 Transportation Policy Plan* 2009, Metropolitan Council Website, Available: http://www.metrocouncil.org/planning/transportation/TPP/2008/G_TransitStandards.pdf (Accessed 24 January 2009), p. G-5

reference real-world system. Keeping the real-world system and this thesis' theoretical model close, also allows for better comparisons to be drawn between the two approaches, by maximising their areas of commonality. With the TransMilenio S.A.'s timetabling practices appearing very sound, there was no reason not to mirror the TransMilenio S.A.'s timetable where-ever possible. This thesis is not, after all, a critique of the existing TransMilenio S.A. timetable but a look at what might be possible, in terms of efficiency, if a specific constraint, namely user comprehension, were to be relaxed.

On the Américas Line, the received timetable data indicates a very heavily-weighted preference towards not running buses any more than five minutes apart, a time that is the same as the Transmilenio's "average non-peak headway" as reported in *Bus Rapid Transit Planning Guide*.[180] In line with the practice of using the existing TransMilenio system as a guide where ever possible, a maximum spread of five minutes between buses was adopted as a hard floor for the timetable development exercise in this thesis. As this thesis' approach to timetabling uses 15-minute time slices, running buses no more than five minutes apart, means having a minimum run frequency level of at least three buses per 15-minutes. Consequently when calculating the total number of runs, if the previously described calculation of dividing the peak passenger load by the capacity of the buses yields less than three buses per 15-minutes minutes, then the total number of runs is set to be three buses. This approach of setting a minimum service frequency, regardless of passenger load, directly

180 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York. p. 767

mirrors the maximum load procedure approach discussed in section 2.4, “An Overview of Transit Timetabling and Scheduling.”

The total number of channels is calculated by dividing the total number of runs, of which there is a minimum of three, by the minimum run frequency level and rounding down. From the data being used, it is possible for this raw calculation to give up to eight channels, but for the purposes of this timetabling exercise the maximum allowed number of channels was set to be four, one all-stop and three express. As has been discussed, this thesis uses the existing TransMilenio system as a guide, and, on the Américas Line, the existing TransMilenio system only allows a maximum of three express channels. Again, the reason for capping the number of express channels at the same number used by the existing timetable was to try and keep what is a highly theoretical timetabling exercise as close as possible to the reference real-world system in cases where doing so does not compromise the fundamental objectives of the timetabling exercise. Keeping the real-world system and this thesis’ theoretical timetabling method close, both better allows for comparisons to be drawn between the two approaches and also facilitates the thesis’ theoretical timetabling approach at least partially capturing some of the resource constraint considerations that informed the development of the real-world timetable, in particular the issue of limited bus-bay capacity of the stops. Bus bay capacity constraints are an important issue generally in transit planning, being discussed in detail in works such as the *Stop, Station and Terminal Capacity* section of the *Transit Capacity and Quality of Service Manual*,[181] and are a key constraint on the TransMilenio BRT

181 Transportation Research Board 2003, *Transit Capacity and Quality of Service Manual (2nd Edition)*, Transportation Research Board, Washington, D.C., U.S.A., pp. 7-i - 7-68

system.[182] Although in theory, the more channels that can be established the better, since more channels mean more stopping patterns and more stopping patterns means a tighter mapping to passenger load, in practice there may well not be sufficient bus-bay resources available at the stops to support a very high number of channels. Consequently, no more than the real-world maximum of four channels, one all-stop and three express, are allowed to be established regardless of whether there are enough runs to support more channels.

Once the total number of runs and channels for the near and far turn-around stops has been calculated, these total levels need to be split between the near and far turn-around stop options. This process is begun by calculating the maximum span load on both the near turn-around part of the line, the spans between stop 0 and stop 13, and the part of the line beyond the near turn-around stop, the spans between stop 13 and stop 17. The concept here is that bus services turning around at stop 13 are able to service spans up to stop 13 but not beyond, and so the distribution of runs between them should be based on this separation. The near and far maximum span loads are calculated by cycling through all spans.

Once the near and far maximum span loads are known, the amount by which the near stop has a peak load level that is in excess of the far stop is calculated. The amount by which the near stop is in excess of the far stop is calculated by subtracting the near load from the far load and dividing the result by the sum of the near and far loads; the result of this

182 From notes taken at a meeting between the author and Alejandro Niño of the TransMilenio S.A. Planning Department on the 7th of June 2006.

calculation is stored in a variable[183] called “excess_near.” The sum of the near and far loads, which are point loads, is being used as a guide to total line load, even though there would be some some load shared between the two point loads. As such, this method of calculating the amount by which the near stop is in excess is likely to somewhat underestimate the degree by which near stop is in excess of the far stop.

It will not necessarily be the case that the near stop load will be in excess of the far stop load, or if it is, that it will be in excess of the near stop load by a sufficient amount to support a new channel. If, rounded down, this number of near-line-part buses is equal to or greater than the minimum run frequency then the near channel is activated by setting a Boolean[184] variable called “near_channel_active” to be true. Otherwise the near channel is set to be inactive.

If the near stop is inactive then the number of runs for the near channel is set to zero. If the near stop is active, then the number of runs for the near stop is set to be total runs multiplied by the excess_near, rounded down. The number of runs for the far stop is set to be the total runs less the number of runs in the near stop. Similar calculations are made to determine the number of channels for the near and far stops. If the near stop is inactive then the number of channels for the near stop is set to zero. If the near stop is active then the number of channels is set on the basis of the proportion of the runs in the near stop. For example, if the near stop had two thirds of the runs it would get two-third of the

183 A variable, in this context, is a named single item, such as a single floating point number, which can be changed by the code without resulting in an error. The term variable is used in contrast to a “constant” which cannot be changed by the code without resulting in an error.

184 A Boolean variable is a variable that can be set to have one of two states, true or false.

channels. If the near stop is active, the minimum number of channels allocated to the near stop is one.

This method of splitting the total number of runs and channels produces good results for the overwhelming majority of the approximately two hundred time-and-direction cases. In running the data however, it was apparent that, due primarily to near channels being allocated which did not improve the situation, some manual adjustment would be beneficial. Four time-and-direction cases had minor adjustments made to them, and two cases had major adjustments made. The exact changes made can be seen in the section labelled “manual override.”

The somewhat ‘rule of thumb’ method employed in doing the run and channel separation calculations, and need to manually adjust the calculation results, both point to the fact that this approach is some distance from being mature or optimal. The primary task of the thesis is to investigate the rapid tracking of passenger load; it is a necessary concomitant of that task that the task of calculating an efficient solution for a single passenger load matrix must also be done. The need to split load between near and far channels is not an essential component of this thesis’ argument, it is only being done because the available real-world data was split in such a fashion. To be able to compare the real-world data with the data produced in this thesis an attempt to reasonably efficiently split the runs between near and far stop, needed to be made. Further refinement, particularly in the direction of simply trying all options and picking the best one, would make for interesting further research.

For the just described purpose of calculating the number of runs and channels, it was most convenient to have floating-point variables storing the number of runs and channels, and to ensure, via the method of calculation, that the number of channels never exceeded four. For the next step, actually determining the express bus patterns, it is more convenient to convert the existing channel data into four explicit channels which are either on of off, and which are either to the near or the far turn-around stops. This conversion is done by establishing two three dimensional arrays, namely “channel_is” which stores a Boolean value recording whether a given channel is on or not, and “channel_type” which can be set to either near or far to specify where the buses for that channel turn around. The three dimensions of both arrays are the direction (inbound or outbound), all time slices, and the four channels. These arrays are filled by cycling through the existing channel data and filling in the newly established arrays. This completes data preparation and pre-calculations tasks.

5.7 Making the Runs: Making Run Groups

Following directly on from the data preparation and pre-calculations tasks, beginning at the in-code comment “exhaust express options”, is the code to generate the groups of express runs for each segment.[185] The basic logic of generating run groups is that each segment is handled independently, and indeed the first version of the code body reflected

185 Each time-slice/direction pair is referred to as a segment. So, for example, the time slice 09:00 to 09:15 in the inbound direction is a segment.

that logic, with a single block of code residing inside a for loop[186] that cycled through all segments.

This implementation, although very memory efficient, in that it had a very small memory footprint, was not processor efficient, in that it required a very substantial amount of combinatorial data to be recalculated for not only each of the 192 segments but also for each of the channels these segments had. A substantial opportunity appeared to exist to significantly reduce processing time, and so this code was re-organised to move the loop for the 192 segments inside the combinatorial loop. Such re-ordering of computational loops is a common practice in programming being discussed in articles such as *How to Optimise Code*[187] and *Optimise Loops*.[188] Although the re-ordering of computational loops required substantial intermediate data to be stored, as the 192 segments were being solved at the same time rather than sequentially, and somewhat reduced the readability of the code, it also reduced the processing time from approximately three hours to approximately five minutes; a very substantial benefit. It is likely that a small further improvement, perhaps a halving or better, of processing time could be achieved by also moving the channels loop inside the combinatorial loop. This opportunity was not pursued as, in the context of this thesis such a

186 A for loop, implemented in the code using the keyword “for”, is used to cycle through a list and to run the same block of code for each element of that list. For example, if one wished to print the academic standing of all students at an institution, one would use a for loop to cycle through every student, running an identical block of code that printed a student’s academic standing for each one. One way to think of the “for” statement is that it allows one to say “for each of these things do this.”

187 *How to Optimise Code* (2008), Programming for Scientists Website, Available: http://www.programming4scientists.com/2008/10/how-to-optimise-code/ (Accessed 19 February 2009)

188 *Optimise Loops*, Aivosto Website, Available: http://www.aivosto.com/vbtips/loopopt.html (Accessed 19 February 2009)

processing time improvement was not seen as being worth further reducing the readability of the code. It should be kept in mind, during the remainder of this section, that the code being described has been partially optimised and so therefore does not always directly reflect the logic of the developed processing method.

The task of making the run groups begins by establishing an array to contain the text feedback which is generated during the process of making the run groups. As, for the reasons just discussed, the segments are being processed as a group rather than sequentially, this array needs to be able to contain text feedback for all segments, which is achieved by making it a two dimensional array, with one dimension being for all time slices and one dimension being for both directions. The variable stored by the array is a string. Text feedback is added to this array during the making of the run groups, and all of this feedback is printed out, segment-by-segment, when all of the run groups have been made. In addition to this text feedback array, a list is established to contain data that will be used to report on the performance of the express pattern selection metric.

The run groups are made by first making the express runs, as necessary, and then by making one all-stop runs for each segment. The groups of express runs for the segments are calculated one channel at a time starting with channel three, the final and therefore least used express channel, and ending with channel one, the first and therefore most used express channel.[189] Immediately before the loop that cycles through the channels is an array called text_feedback which is used to store feedback information generated during the process of making the

189 The all-stop channel is numbered channel zero.

run groups. This feedback is not printed-out until after the entire run group generation process has ended because, due to the previously discussed efficiency focused code re-sequencing that has been conducted, useful partial data is not available.

Immediately inside the loop that cycles through the channels is a block of code that prints whether the channel is a near turn-around stop channel or a far turn-around stop channel. Following on from that is a block of code that calculates the number of all-stop buses that would be required to service the passenger load, for each segment that has any active channels. This value is calculated primarily to rapidly exclude any express run solution that, in conjunction with the all-stop buses, uses more runs than than a straight all-stop solution alone. For example, if a given passenger load could be serviced using say 15 all-stop buses then any combination of all-stop and express buses that uses more than 15 buses, say eight all-stop and eight express buses, is excluded from consideration.

Having set a ceiling on the combined total of all-stop and express runs, the primary data stores to be used during the process of searching for an ‘optimal’ express group are declared. The central method used in searching for an ‘optimal’ express group is to examine a large number express group options, comparing each to the currently found best option, and, if the current express group option is better than the existing best option, replacing the existing best option with the current express group; when there are no more express group options to test, then the existing best option can be used as the overall best option. As such, it is necessary to store enough data about the best option both to do an

effective comparison with other candidate best options, and when the candidate best options have been exhausted, to report all necessary data on the final best option. To meet these objectives five arrays are declared, namely best_express_run_group_found, best_express_run_group_stops, best_express_run_group_links, best_express_run_group_number_of_runs and best_express_run_group_metric. As all segments are being calculated at the same time, these arrays are two dimensional arrays, with the two dimensions being the time slice index and the direction.

The best_express_run_group_found array contains a Boolean value that records whether a best express run group was found. It is possible for the combination search system to not find an acceptable express combination, for reasons that will be discussed shortly, and this Boolean value allows for this possibility. The best_express_run_group_stops array contains a vector of stop indexes, in the form of integers, which are the stops the current best express run group stop at. The best_express_run_group_links array contains a list of link indexes which are the links between the stops, consequent from the stops that are being stopped at and the direction. Although this data could have been derived from the stops and direction data it was both more convenient, and more computationally efficient, to have this data ready-to-hand. The best_express_run_group_number_of_runs array contains the number of runs, per 15-minute period. The best_express_run_group_metric array contains the metric that is used to assess whether one candidate run group is better than another candidate run group.

As discussed in section 4.6, “An Overview of an Ultra-efficient Timetabling Method”, the metric that has been chosen was minimum total bus run time. The initial ‘naïve’ implementation of a minimum total bus run time metric was implemented exactly so, with the bus stopping pattern that minimised end-to-end travel time being favoured. Unfortunately, because ‘filling’ a bus with passengers is an inherent part of choosing a stopping pattern, and because passenger boarding and alighting times form part of total bus run time, there was a tendency for this metric to eschew any pattern that involved picking-up and dropping-off a lot of passengers. In other words this initial ‘naïve’ metric implementation led to buses avoiding servicing short passenger trips, as those trips only helped fill the bus for a short while, while giving a full measure of board and alight time. If it were the case, globally, that the picking-up and dropping-off of passengers was avoidable then this avoidance behaviour would have been fine, but it is the case that such work can only be avoided locally; if one bus does not pick up a passenger another one must. It is also the case that if all run groups for a time period were being calculated simultaneously then this behaviour would not matter, as then it would not be possible for difficult-to-service trips to be passed on to a latter calculation phase. As will be discussed shortly, though, sequential optimisation calculations are made, and so such behaviour is, to say the least, highly undesirable.

This issue, identified as the risk of being caught in a sub-optimum, was discussed in detail in section 4.6, “An Overview of an Ultra-efficient Timetabling Method.” The specific risk this section identified was that an early stage of the optimisation sequence would select an express pattern

which made it impossible for later stages to find their own efficient express pattern. This risk has indeed eventuated with the first implementation of the metric, leading to highly sub-optimal solutions as a result of early stages selecting all of the easy to service loads, and leaving all of the difficult to service loads to later stages. Consequently, the first implementation of a minimum total bus run time metric, turned out to be unusable in practice, as it yielded highly sub-optimal solutions when used in conjunction with sequential calculations.

To capture the essential character of the desired metric of minimising total bus run time, in a manner that would allow sequential calculations to be conducted, a metric that was not as tightly bound to the passenger data was required. In pursuit of this goal, the second metric that was tried was one based on minimising the total number of intermediate stops that buses stop at. As both the passenger load level and the bus performance characteristics are invariant, it is the case that the only way for a bus to get from one end of the line to the other faster is to stop at less stops. By better capturing the essential aspects of the problem, it was hoped that this second metric would not exhibit the short-passenger-trip avoidance behaviour of the first ‘naïve’ metric. In use, this second metric did not exhibit the poor behaviour of the first, and so was adopted.

The development of a metric, on the second try, that did not result in early stages taking all of the easy to service loads, provides an intuitive demonstration that the risk of being caught in a sub-optimum has been contained. Although there is no reason to believe that the solution selected by the adopted metric will be the optimal solution, the specific area of sub-optimal risk has been identified and addressed, and so it is

likely that the selected solution will be one of the set of ‘good’ solutions. Just how close the selected solution is to the optimum solution could be assessed by exhaustively searching all possible solutions for the two express pattern case. In section 4.3, “A Practical Characteristic – Computational Optimisation” it was shown that exhaustively searching all possible solutions for the three express pattern case would take approximately 40 years on the specified ‘normal’ computer. The two express case, though, requires approximately 65 thousand times less calculations than the three express case, and could be computed in approximately five hours. This calculation was not conducted as by the time this issue with metrics and sequential calculations had been clearly identified, the structure of the software was settled and significant re-working would have been required to do such an assessment on the performance of the second metric.

This second metric was used to judge one option against against a previous option, so as to be able to select the better of the two options. As previously discussed five data stores were used during the process of searching for an ‘optimal’ express group, namely best_express_run_group_found, best_express_run_group_stops, best_express_run_group_links, best_express_run_group_number_of_runs and best_express_run_group_metric. Any method that involves comparing one element with a previous element requires default values to be set, otherwise what is the first element is a set of elements to be compared to. In this case a default value only needed to set for the best_express_run_group_found and the best_express_run_group_metric arrays; if an appropriate express bus run cannot be found then the

default response is to run all-stop rather than express buses, making the three express bus data arrays redundant. The best_express_run_group_found array is set to false and the best_express_run_group_metric array is set to the minimum number of all-stop runs required to service the segment's load multiplied by the number of intermediate stops.

It may seem that the selected default metric is a mere place-hold, one put there just to get the 'optimisation ball rolling', as a naïve assessment would say that any express choice is going to be better than one using only all-stop buses. This is not the case. Although any combination of express runs and all-stop runs that uses the same total number of buses will have a better metric score than the all-stop option, it is not necessarily the case that there will be such an option available. In splitting the load between two groups of runs, an express and an all-stop group, it may be the case that one more total run is required. For example, a load that, say, seven all-stop buses could service, might be better serviced by 3.5 all-stop runs and 3.5 express runs. It is not though possible to have 3.5 runs; a run is a bus and it is only possible to have three or four buses, so the a load that required 3.5 buses to service it has to be serviced using four buses. Consequently, for this example, a load that could be serviced by seven all-stop runs, requires eight runs when serviced by all-stop and express buses. By adding a full extra run, the improvement in the metric score achieved by the express runs stopping at less stops may be more than negated by stops added by the extra run, and so, even in cases where there is sufficient load to activate express channels, using express buses might not be the most efficient option.

This is an example of what is known as the integer problem, situations where only whole-number solutions are feasible.

With the data stores needed to find the most efficient run option declared and the necessary default values for these data stores set, the process of cycle through and judging all combinations to the identify the most efficient option can begin. How this is done, and consequently all of the code in the next code block, which begins at the label “find the best express run group”, is the subject of the next section, “Making the Runs: Judging a Single Express Run Group.” On exiting this code block, at the label “run the best express run group”, all combinations have been judged and the “best express run group” variables now contain the data for the express run group judged to be the best.

Once the best express run group has been found it is necessary to convert the run data from its present form, best for doing the optimisation search, to an output data form, best for running analysis on. The task of doing this data conversion is handled by a function called make_one_run_group. The reason that this data conversion task has been moved into a function is that this task is performed twice, once when the express runs are generated and once when the all-stop buses are generated. Of the variables used to calculate the best express, namely the stops, links, number-of-runs and metric, all but the metric is passed to the make_one_run_group function; with the metric having played its role of identifying the best express pattern it is not needed further. Along with this information, the make_one_run_group function is also passed the segment index, the direction and time slice, the loads, the previously

calculated links_loads and, for output purposes, the list of runs and the previously declared output text string.

The first task of the make_one_run_group function is to isolate those loads that are being serviced by the given express pattern and to scale them down to one run. This is done by establishing and then zeroing an array whose dimension is all-links, and then, for each of the given stopping pattern’s links, copying across the passenger load divided by the number of runs. The single runs are generated by cycling through the number of runs. For each run, a datum time for the run needs to be calculated, which as has been previously discussed is the centre of the line, with the load having been adjusted to match this datum. For each 15-minute period there are a given number of runs that need to be evenly spread across the 15-minute time period. As load has to build before it can be cleared the runs are made to be flush with the end of the 15-minute period; so, for example, if there were three runs between 06:00 and 06:15, those runs would be at 06:05, 06:10 and 06:15.

With the passenger load isolated and the time of the run calculated, all of the information necessary to construct a run is available. The information used to make a run is the direction of the run, the time, the load time duration that the run covers, the stops that the run stops at and load for one run. Once constructed the run is placed in the list of runs and text feedback is made. The text feedback consists of the start and end time of the run, the number of runs in the group, the duration of the run, the average occupancy as a percentage, the peak load and stops that are stopped at. To finalise the process of making one run group the passenger loads on the links serviced by the run group are zeroed.

Having made the runs, the channel guides, the record of the current number of channels and the current number of runs, need to be updated. As previously discussed, due to rounding issues, there is a small amount of play in the number of runs that may have allocated to a given run group. Consequently, a record of the current number of channels and the current number of runs needs to be kept so that the target number of runs for a given channel can be assessed afresh for each channel as they are reached. The number of channels is updated by subtracting one channel, the current channel, from the number of channels. The number of runs is updated by subtracting the number of runs in the just calculated channel. This completes the calculations required to run the express runs.

After the express runs have been made, an all-stop run group is always made; the all-stop channel is channel 0 by definition. The all-stop run group is made using the same make_one_run_group function that was used to make the express run groups. All remaining runs are run as all-stop runs. After the all-stop run group has been made, the channel guides are updated by decrementing the number of far channels by one and setting the number of runs in the far channel to zero. The updating of the channel guides is not strictly necessary as the all-stop run group is the last run group to be made, but keeping an accurate track of the channels and runs right until the end allows a check to be made as to whether all the channels and runs have been used. As it is the case that the normal operation of the program should lead to all of the runs and channels being used, the checking as to whether this is the case is done

using an assert[190] statement. The process of generating the runs is finalised by printing out both the text feedback and the data about the performance of the express pattern selection metric, and also writing the generated runs out to a file.

This section has addressed the seventh research question of this thesis, namely whether a simple and effective metric to judge the efficiency of BRT express service stopping patterns can be easily devised. This question has been answered in the negative. Although a simple and effective metric to judge the efficiency of BRT express service stopping patterns can and has been devised, this section has shown the significant difficulties involved in the development of such a metric.

5.8 Making the Runs: Judging a Single Express Run Group

The previous section worked through the processing sequence used to determine the run groups that were to be run, with the exception of the code that actually cycles through the combinations and judges the express run group options. This code body, marked with the label “find the best express run group” was treated as a black box,[191] with the passenger load data and default best express run group data being fed in the top and the final best express run group data ‘magically’ appearing out the bottom. This section ‘opens up’ this black-box and discusses in detail the mechanics of how the combinations are cycled through, and how each combination is judged.

190 An assert statement terminates the program execution if a condition given to it is not met, in this case the condition is that the channels/runs should be empty/zeroed. Assert statements are usually used in cases where an error is not expected; as such they are a double-check rather than a normal part of error reporting.

191 The term black box here refers to a code body whose general function has been described, but whose specific operation has not been described.

To cycle through the combinations a utility function called next_combination_all is used, which performs the necessary mathematical operations to cycle through all of the combinations for a given channel in a logical manner. As the implementation of a combinations algorithm is not a topic of this thesis the operation of this algorithm will not be discussed, beyond the fact that the algorithm was successfully tested before use. For the purpose of this timetabling exercise the end stops are always stopped at, and only the intermediate stops given to the next_combination_all function to be cycled through. To operate the next_combination_all function requires a list of all of the intermediate stops to be passed to it, and so the “find the best express run group” code body begins by calculating the intermediate stops for the near and far turn-around stops and storing them in an array called all_intermediate_stops.

The way in which the next_combination_all function works is that it will only iterate over all of the combinations for a given number of stops. So the next_combination_all can be used to iterate over all of the combinations that involve stopping at one stop, or alternatively two stops, or alternatively three stops, but will not automatically iterate over all possible combinations. For this optimisation exercise all possible combinations are to be checked, and so all of the possible stop levels need to be manually iterated over. This manual iteration is performed by a for loop which declares the variable combination_stop_level. The final component that the next_combination_all function needs to operate is a list containing the initial combination for each combination stop level, with the initial combination simply being that, for a given stop level, the

earliest possible stops are stopped at. So, for example, if the stop level is two — two stops are being stopped at — then the initial combination is to stop at the first and second stop. This initial combination list is declared as combination_intermediate_stops immediately inside the for loop that sets the combination stop level, and is filled with the appropriate initial combination data as just described.

The reason that the list of initial combination stops is declared as combination_intermediate_stops not initial_intermediate_stops is that it is this list that the next_combination_all function operates on, advancing the combination in combination_intermediate_stops to the next logical combination each time it is called. The advancing of the combinations is handled inside a do/while loop[192] with the next_combination_all function being called as part of the while test, a test that fails when the next_combination_all function has provided all combinations. The code inside the do/while loop is sequentially presented with all the possible stopping combinations for an express service, allowing it to judge each one in turn and to update the previously discussed four “best express run group” variable if the combination under review is better than current best combination.

The first task performed inside the do/while is to report how far the processing task has progressed to the user. What with there being 65,536 combinations, 2 to the power of 16, the processing takes an appreciable amount of time.

192 A do/while loop executes a block of code then checks a specification condition and if that condition is met executes the block of code again; a sequence that continues until the specified condition is not met.

The next task is to make an array, called combination_all_stops, by copying in the intermediate stops, as set by the next_combination_all utility function, and then adding the first and last stop; the stops in the combination_all_stops array are the stopping pattern for the current combination. Next the links for the stopping pattern are determined and stored in an array called combination_links. As a channel can be either a near or a far channel the combination_links array has a near/far dimension. As each single combination is a valid stopping pattern for both the inbound and outbound direction the combination_links array also has a direction dimension. Finally, the combination_links array has a third dimension, a negative/positive dimension, whose purpose is to allow both the links delineated by a given combination to be stored and also, separately, for those links that are not delineated by a given combination. The reason for recording the links that are not delineated by a given combination is because, as all links must be serviced, any link that is not serviced by the express runs represented by a given stopping pattern needs to be serviced by all-stop runs. The negative could also be thought of as the remainder; the links that are left to be serviced after the express runs have serviced their links.

The combination_links array is filled by cycling all from-stops and all to-stops, with the link that is represented by a particular from and to stop pair being added to the list of links stored by the combination_links array. The near/far dimension of the array is set by cycling across the near and far options simply by using a for loop. The direction dimension is set by assessing whether the to-stop is before or after the from-stop on the line. The positive/negative dimension is set to be positive if both the from and

to stop are present in the combination_all_stops array, otherwise it is set to be negative.

Next, if there is a channel active for this segment and channel level, the process of comparing the metric for the current combination to the metric for the current best option is conducted. This process begins by calculating how many runs would be required to service the passenger load on the links which result from the current combination. The number of runs is calculated by rounding-up the peak load divided by the bus capacity, and if this figure falls below the minimum runs allowed setting the number of runs to be the minimum allowed. The number of runs needed to service a given stopping pattern may well not be the number of runs that are sought after for the channel, a magnitude that is calculated next by dividing the current number of runs by the current number of channels. This calculation can and often does yield a non-integer result. As runs must be an integer value either the lower or higher integer value is viewed as being an acceptable number of runs for this channel. If the number of runs actually needed to service the load that is consequent of the current stopping pattern matches either the higher or lower integer needed-runs value, then the combination is further assessed to see if it superior to the current best combination, otherwise it is discarded.

A given express stopping pattern will service certain links and leave the remainder unserviced; these remaining unserviced links will have to be serviced by all-stop buses; a ‘good’ stopping pattern is one where the number of middle stops of the express runs and the ‘remaining’ all-stop runs combined is low. Consequently the first task is to calculate the number of all-stop runs there will be when the load for this combination

of expresses is removed. This calculation is made in the same manner as for the express buses — in short, rounding-up the peak load divided by the bus capacity — but using the previously calculated negative links. The final metric, for which to compare with the current best metric, is obtained by multiplying the number of express runs by the number of intermediate stops for that express run and adding that number to the all-stop runs multiplied by the number of intermediate stops.

If this number is lower than the current best express run group metric this combination is the new best express run group and the data being stored about the best express run group is updated. With all combinations being tested, after the last time this code block is run the best combination out of all of the combinations will be stored as the best express run group. The remaining code before the end of the do/while loop that is generating the combinations is housekeeping; data is gathered which will later be used to assess the performance of the express pattern selection metric, a combination counter is advanced to enable visual feedback to be given as to how many combinations have been processed to date, and the near turn-around combination is also advanced. It is necessary to advance the near turn-around combination as, unlike the far turn-around combination it is not being advanced as a part of the do/while loop. With these housekeeping tasks complete, the process of assessing the combinations is finished.

5.9 Deadheading the Runs

The previous function, “make run groups from loads” discussed in sections 5.6, 5.7 and 5.8, generated services to meet the load on the line, one segment at a time without regard as to whether those services matched up to form continuous runs that begin and end at the outermost stop. The method to be used to form the runs up into continuous runs is to add deadhead runs, runs that do not carry passengers and that do not stop at stops. Menu option “e” of the Matilda software, “deadhead runs”, which can been seen near the centre of the Figure 4.7 data-flow diagram, performs this task of adding deadhead runs to the previously calculated all-stop and express runs such that all bus runs start and end at the outermost stop.

At first glance deadhead runs can seem wasteful as the role of buses is to move passengers, not to to travel the line empty. It is, though, a design principle of this software that there is no point trying to pretend load imbalances don’t exist when they do. The reality of load imbalances is the reason that deadheading is practised, and that issues relating to deadheading practice are researched in papers such as *The Real-time Deadheading Problem in Transit Operations Control.*[193] The Matilda deadheading algorithm does try to minimise the number of deadhead buses, by pairing as many inbound buses with outbound buses as possible, before resorting to adding deadhead buses.

193 Eberlein, X. J., Wilson, N.H.M. and Barnhart, C. (1998), *The Real-time Deadheading Problem in Transit Operations Control*, Transportation Research. Part B: Methodological, Volume 32, Number 2

Before the DeadheadRuns function itself there are two utility functions, the first called less_than_outermost_FO[194] and the second called do_pair. The less_than_outermost_FO function implements a secondary way of sorting runs. The primary way of sorting runs, as implemented in the run class[195] itself, is to sort the runs based on the time they are at the innermost stop, which is the arrival time at the innermost stop for inbound runs and the departure time at the innermost stop for the outbound runs. The secondary sorting method, implemented by the less_than_outermost_FO function is to sort the runs based on the time they are at their outermost stop, which is the departure time at their outermost stop for inbound runs and the arrival time at their outermost stop for the outbound runs.

The primary sorting method on the innermost stop is very simple; however two issues complicate the secondary sorting method. Firstly, all runs have the same innermost stop, stop 0, whereas a run can have one of two outermost stops, the turn-around stops, stop 13 and stop 17. Although this choice of two outermost stops does not complicate the implementation of the secondary sort method as all runs can be asked what their outermost stop is, care must be taken not to use the secondary sort method on a list of runs with a mix of outermost stops, or the resulting sort order will be of no real meaning. The second issue which complicates the secondary sorting method is that there is a time delay at

194 less_than_outermost_FO is technically a function object not a function, thus the FO designation. The difference between a function and function object is not material here.

195 A class is a logical representation of a clearly delineated concept or thing. In this case, the concept “run” is clearly delineated and a run class has been written to store run information such as the direction of a given run and the stops that that run stops at. Classes can also specify a default sorting method that can be applied to a list made up of elements of that class; in this case such a sorting method would enable a list of runs to be sorted.

the outermost stop, whereas there is not one at the innermost stop. It is commonplace with bus timetables for time to be added somewhere in circuit to allow timetables to be kept to even when the run takes slightly more time than expected, due to traffic conditions or boarding and alighting delays. For buses that were turning around at the innermost stop on the Américas Line, the TransMilenio S.A.'s timetable adds an approximate one and a half minute hold at the bus's outermost stop. No delay is added at the innermost stop, presumably due to the limited space available at the innermost stop and the ample space available at the outermost stops, particularly the far outermost stop, stop 17, which is a bus terminal rather than bus stop. As previously discussed, the timetabling method in this thesis mirrors the TransMilenio S.A. timetable wherever possible, and so a one and a half minute hold time is added to the bus's outermost stop, when they are being sorted on the outermost stop, which stops a bus leaving on a new run before its hold time is completed. In the less_than_outermost_FO function the hold time can be seen being added to the end of the outbound run by the setting of a flag[196] on the call of the "Time" function.

The do_pair function is used by the DeadheadRuns function, which uses it to pair runs at both the innermost and outermost stops. As parameters, the do_pair function takes the list of runs to be paired, a parameter to specify whether the pairing is occurring on the innermost or outermost stop, and a time parameter which specifies how far apart in time two runs are allowed to be and still be paired by the do_pair

196 A flag is just a variable that reports the state of something, commonly implemented as a Boolean. In this case the flag reports of whether the hold time should be added or not added.

function. Runs too far apart in time are not paired, otherwise, for example, a morning run might be paired with an afternoon run, and consequently a bus could be waiting around for eight hours. The exact amount of time for which a bus may be held to be paired with another bus, and the reasons behind those times, are detailed shortly.

The first task of the do_pair function is to sort the runs that have been passed to it. If the runs are being paired on the innermost stop then they are sorted on the basis of the time they arrive/depart the innermost stop, and if the runs are being paired on the outermost stop then they are sorted on the basis of the time they arrive/depart the outermost stop. The purpose of this sorting is to allow pairs of runs which are adjacent in terms of the time that are at the innermost or outermost stop, to be readily identified. Adjacent stops may be nested inside of one another; for example, three inbound buses may arrive at the sort stop followed shortly after by three outbound buses leaving. The method used to match such nested pairs is to identify, match and remove the innermost of pairs, and then to iteratively repeat this process. After the initial pair has been removed, what were the two surrounding stops will now be the innermost pair and so removable using the same method.

As the number of nested pairs is unknown, a do/while loop is used to repeatedly execute the process of identifying, matching and removing run pairs. A flag called found_pair is set to be false at the beginning of the process and is set true when a pair is found, so that the while statement will continue to execute for as long as further pairs are found.

The pairing process begins with all the runs being cycled through. For each run, the run after it is identified, with the current run being

called run_current and the run after it being called run_next. It is then determined whether these two runs are pairable. For the innermost stop, the two runs are pairable if the current run is inbound and the next run is outbound. For the outermost stop, the two runs are pairable if the current run is outbound and the next run is inbound. If the two runs are pairable a test is then made to see whether the runs are close enough together in time, as specified by the time parameter passed to the do_pair function, to be paired. If the two runs are close enough together in time then their runs are marked as paired and the found_pair flag is set to true. When all the runs have been checked to see if they can be paired, the runs that were paired are filtered out from those that remain unpaired and, if at least one set of pairable runs was identified the process is repeated. Once no sets of pairable runs are identified the paired runs are recombined with the non-paired runs and the do_pair function ends.

The DeadheadRuns function processes runs, and so the first task this function performs is to read in the runs produced by the previous MakeRunGroups function. After the runs have been read in, the do_pair function is used to pair all the runs that can be paired at the innermost stop. The maximum allowable hold time for a pairing to occur is the time that it would take a deadhead bus to travel from the far outermost stop to the innermost stop, a time duration that is calculated on the fly by constructing such a run and asking it how long it takes. The logic behind this choice of hold time is that sending a deadhead bus effectively takes that bus out of service for the duration of its deadhead run, and so any

hold time less than the deadhead run time is preferred over a deadhead run.

If the hold time would be greater than the deadhead run time, the time that a bus is out of service is minimised by sending a deadhead bus, and those runs that were not paired by the previous do_pair function call are ‘paired’ with a deadhead bus. Deadhead buses are constructed by constructing a match ‘mirror’ bus to the service buses. Unpaired inbound service buses are paired with outbound deadhead buses that leave the innermost stop at the same moment the service bus arrives. Unpaired outbound service buses are paired with inbound deadhead buses that arrive at the innermost stop at the same moment the service bus leaves. Although some service buses go back to the near outermost stop and some service buses go back to the far outermost stop, at this stage the deadhead buses are only constructed going back as far as the near outermost stop. As going back to the near outermost stop is a shorter run than going all the way back to the far outermost stop, all opportunities to only deadhead buses to the near outermost stop are assessed first. When all the deadhead runs have been constructed they are placed in the list that holds the service runs, and all of the runs in this list are marked as unpaired.

At this stage all buses are paired at the innermost stop, in that if a bus arrives at the innermost stop, it will have a scheduled run allocated to it which will have it leaving the innermost stop after a relatively short period of time. Consequently, with the objective being to connect the runs back to the far outermost stop, any run that goes between the innermost and far outermost stop is now done; with all the loose ends now having

been moved to the near outermost stop. The loose ends at the near outermost stop are ‘tied up’ in a similar fashion to the loose ends at the innermost stop, using the do_pair utility function, but in this case with the function being told to pair the runs on the outermost stop. The maximum hold time given to the do_pair in this case is based on the same reasoning as for pairing on the innermost stop, but is calculated slightly differently.

The reasoning when pairing on the innermost stop was that a bus should wait up to the time that a deadhead run would take; with the deadhead run in this case being an added run operating between the innermost and far outermost stop. In the case of pairing on the near outermost stop, if a pair cannot be found then further buses will not be added and the existing bus run will be extended back to the far outermost stop. As such, the effective added deadhead run is from the near outermost stop to the far outermost stop and back again, which, as this run passes the outermost stop must include the run hold time of one and a half minutes. As with pairing on the innermost stop, the hold time is calculated on-the-fly by constructing a deadhead run and asking it the length of its run time. The total hold time was calculated as the time of the deadhead run between the near outermost and far outermost stop multiplied by two plus the slack time.

As with the previous pairing, not all runs will be paired, some runs may be too far apart in time to pair, and also there may well be an overall imbalance in the total number of runs inbound compared to the total number of runs outbound, giving runs that are unpairable no matter how far apart they are allowed to be in time. The runs that are not paired by

the do_pair function are extended back to the far outermost stop. Some of the runs that are to be extended back to the outermost stop are deadhead runs; these runs are extended back by erasing current lists of stops the buses stop at and adding the innermost and far outermost stop to that list. Some of the runs are service runs; these runs are extended back by adding the far outermost stop to the list of stops the run stops at. This completes the deadheading, so the (previously) unpaired_runs are amalgamated with the completed_runs and the complete_runs then written out.

With the runs now deadheaded, all of the runs that leave the far outermost stop return to the far outermost stop, and when turning around at either the innermost stop or near outermost stop only wait for a limited amount of time. Consequently, the runs are continuous and the ‘state’ of the transit line is the same when the line closes as it was when the line opened, specifically all of the buses start and end the day at the far outermost stop. By deadheading the runs meaningful analysis can now be conducted, in particular with regard to the total number of buses that are needed to service the line. This analysis is conducted by the function RunAnalysis, discussed in section 5.13, “Analysing the Runs.”

5.10 Making the Passengers from the Runs

Passenger load and passengers trips, although closely related, are not quite the same thing; load is a requirement to travel whereas trips are actual journeys on, in this case, a bus. The primary data analysis system dealing with passengers in Matlida analyses trips not load. Passenger trips are the primary form of data analysis because by

analysing trips useful information can be gained not just about how many passengers were moving around the transit line but also how fast they were moving around the transit line. The previously discussed combinatorial optimisation code that tailored a set of runs for each segment, allocated the passenger load associated with each run to that run. By knowing the load there is on each run, passengers records can be generated for the purpose of later feeding those records to the passenger data analysis code.

Menu option “f” of the Matilda software, “make paxs from run groups”, which can been seen at the centre right of the Figure 4.7 data-flow diagram, calls a single function, MakePaxGroupsFromRuns, which takes the passengers load data associated with each run and converts it into passenger records. This function begins by reading in the runs and declaring a list of passenger groups, called paxs, to hold the passenger group records to be made. The paxs list is then passed to each run using a member function called MakePaxs, which generates the run’s passengers and adds them to the list of passenger groups.

The MakePaxs function cycles through all links, and for any links that which do not have a zero load it makes a passenger record. The number of passengers for the passenger record is the magnitude of the load. The arrival time at the departure stop for the passenger record is set to be the departure time for the trip minus the average time spent waiting at the departure stop. The link for the passenger record is set to be the current link. The departure and arrival time for the passenger record is set to be the departure and arrival time of the run from the from and to stops of the current link.

Once all of the passenger records have been made, the records are written out for future use by the passenger data analysis function.

5.11 Making the Passengers from the Loads

The Matilda software can perform complex data analysis on two types of data, namely runs and passengers. When generating the ultra-efficient timetable for this thesis, runs are produced and passengers loads allocated to those runs which allows, as detailed in the previous section, passenger records to be generated. Having both run and passenger data available allows not only the run and passenger data to be analysed in isolation, but also for analysis that correlates the ultra-efficient passenger and run data to be conducted. In itself, such joint passenger/run analysis of the ultra-efficient timetable data is of significant utility, but far greater insight could be gained by comparing this joint passenger/run analysis of the ultra-efficient timetable with a similar joint passenger/run analysis for the efficient timetable. The run data for the efficient timetable was provided by TransMilenio S.A. and converted into a format analysable by the Matilda software as described in section 5.4, “Converting the TransMilenio Run Files.” Passenger data for the efficient timetable is not directly available; load data was provided by TransMilenio S.A. but as has been previously discussed, load data is not quite the same thing as passenger data. Of particular relevance here is that load data does not know when it was serviced, whereas passenger data does.

The fundamental method used to convert the efficient timetable load data into passenger data is through the application of average wait and

transit time data. This method yields data that is accurate but not precise with regards to its time data. As this thesis is examining the differences between efficient and ultra-efficient timetabling, the differences in the data magnitudes are expected to be non-large in many cases, and so data that was precise would be required to do reliable comparisons. Consequently, no direct comparisons are made in this thesis between the ultra-efficient and constructed efficient passenger data in areas where the time data is central. The ‘constructed’ passenger data is only used to ‘place’ the passenger loads on the line at approximately the right time and for approximately the right amount of time.

Menu option “g” of the Matilda software, “make paxs from loads”, which can been seen at the top right of the Figure 4.7 data-flow diagram, performs the conversion of load data into passenger data, thus allowing comparisons between the ultra-efficient timetabling method and efficient timetabling method passenger/run data to be conducted. A single function, MakePaxsFromLoads, is called. This function begins by defining the average wait time and the average line travel time for the TransMilenio, as timetabled by the TransMilenio S.A. By reviewing the TransMilenio S.A. load files reasonable approximate values for the average wait time and average line travel time were determined to be one minute and 30 minutes respectively. These two reference times are declared at the start of the MakePaxsFromLoads function.

To begin the process of converting the load data into passenger data, the loads are read in. The load file records the from and to stop, whereas passenger records are defined using a link index, and so next an array called link_loads is established, which stores the loads based on a link

index, and the loads are moved into this new array. With the load data in the correct format, constructing the passenger records is done by simply cycling through all time slices and then all links. For any time-slice/link pair where there is load, a passenger record is created. To create a passenger group record requires five pieces of information about the passenger group, namely its size, departure stop arrival time, link, departure time and arrival time. The size of the passenger group is set to be magnitude of the load. The departure stop arrival time is set to be the centre time of the time slice minus the average wait time. The link index is directly settable as the load has been moved into being accessed via its link index. The departure time is set to the middle of the time slice and the arrival time is set to be centre time of the time slice plus the estimated time taken to travel the line. The time taken to travel the line is calculated by dividing the length of the trip by the total line length and then multiplying the resulting number by the estimated total line travel time. Once all the time slices and links have been cycled through, and all of the passenger groups thereby created, the passenger groups are written out to a file for later analysis.

As previously discussed, due to the manner the passenger groups have been created here, using average time magnitudes, care needs to be taken to limit the types of data comparison that are conducted. The function that produces the data used to conduct analysis on the passenger data, discussed in the next section, generates a wide spectrum of analysis data, as the ultra-efficient timetable passenger data does not have the same limitations of this ‘constructed’ passenger data. In the next data analysis chapter, though, it can be seen that, when dealing with

this constructed passenger data, restricted use is made of this constructed data. In particular, no direct analysis is made of the wait time or line travel speed performance, or related metrics, for this constructed data. The objective here has only been to produce passenger data that places the load on the line at approximately the right time for approximately the right duration, thereby allowing the key propositions of this thesis to be tested.

5.12 Analysing the Passengers

The Matilda passenger data to date has been created and stored as a list of passenger group records, with each passenger group record storing five pieces of information; group size, departure-stop arrival-time, link, departure time and arrival time. At the end of every function that creates passenger records, those records are written out to a file in a human-readable text format, and, in principle at least, this file could be used as the source of the passenger data for the purposes of data analysis. It is the case, though, that the format of the data in the passenger data file was designed to be the best format for the software to manipulate, without regard to whether this data format would be useful for other purposes. The format of the passenger data, as is, is not amenable to being analysed, producing graphs and such like. This section describes the data manipulation that is conducted to produce data in a format amenable to data analysis.

The primary data conversion that needs to be conducted to make the passenger data more amenable to data analysis, is to group the data into time slices. Such grouping allows a view of what is occurring, vis-a-vis for

example load on the line, at a given time or across a given time period during the day. The time duration granularity that has been used throughout this thesis is fifteen minutes; the load data received from TransMilenio S.A. was in fifteen minute increments and consequently the run data constructed as part of this thesis was also constructed at a granularity of fifteen minutes. For the purpose of data analysis this time duration granularity has been continued, with the analysis data being grouped into fifteen minute time slices. A secondary data conversion that will also be conducted to make the data more amenable to data analysis is to present the data both separated into inbound and outbound groups and as a joint bi-directional group. These two groupings, by time and direction, allows, for example, one to say that from 09:00 to 09:15 on the inbound line, the, for example, line load was at such-and-such a magnitude.

Menu option “h” of the Matilda software, “produce pax group analysis data”, which can been seen at the bottom left of the Figure 4.7 data-flow diagram, performs this grouping by time and direction by calling one function, namely PaxAnalysis. This function begins by reading in the passenger data. It then passes this passenger data to the same function, AnalysePaxs, three times; once to compile data on the inbound direction, once to compile data on the outbound direction and once to compile data on both directions at the same time.

The AnalysePaxs function begins by declaring three string arrays, one to store compiled time data, one to store speed data and one to store distance data. Each of these arrays has one dimension, with the

dimension being an enumerated type[197] called ETripTimeType. The concept of a trip-time type is that any given trip has three time durations associated with it; the time a passenger has to wait for their service to arrive, the time the passenger's trip takes from boarding to alighting, and the total time that a passenger's trip takes from when they arrive at a stop until they alight at their destination stop. The first time, the time a passenger waits for their service, is referred to as the wait time. The second time, the time the passenger's trip takes, is referred to as the transit time. The third and final type of time, the total time that a passenger's trip takes including the wait time, is referred to as the journey time, and is the sum of the wait and transit times.

Although the three arrays for time, speed and distance each have the same space for wait, transit and journey type data to be recorded about them, not all of these types are equally applicable with regards to recording time, speed and distance data. For example, reporting speed calculated using the transit and journey times has utility, but it is not clear what a speed calculated as the distance divided by the wait time might mean. Also, with the distance not being dependent on time the distance for all three time type will be identical. To simplify processing, data without a clear meaning and redundant data, as just described, is calculated but not written out to a file.

Before the data is compiled to fill the time, speed and distance arrays, a header line which specifies whether the data is wait, transit or journey time type data is stored to avoid later confusion. To actually fill

197 An enumerated type is an enumeration of discrete categories. For example, one might have an enumerated type called shapes, that has had three types defined, say triangle, circle and square.

the time, speed and distance arrays, each time slice is cycled through, and for each time slice each trip-time type is cycled through. For each time-slice/trip-time-type pair, time, speed and distance data is calculated.

These calculations are begun by declaring and zeroing two variables, time_slice_distance and time_slice_time, which record the total time taken and total distance travelled for the passengers. To calculate these two variables, all of the passenger records are cycled through. For each passenger a test is made to see if the passenger is travelling in the desired direction, which can be either inbound, outbound or inbound-and-outbound. If a given passenger is not travelling in the desired direction they are not considered further. Next a test is conducted to see if the relevant time duration — wait, transit or journey time — for the passenger overlaps the current time slice for which data is being compiled; in the normal case only a small percentage of passengers will overlap.

If a passenger is travelling in the desired direction and there is some time overlap, then an overlap ratio is calculated by dividing the time duration of the overlap by the total time for the passenger. This overlap ratio is used to scale the total distance travelled by a passenger, thereby allocating the distance travelled among the time slices in proportion to the degree by which they overlap a time slice. Specifically, to calculate the distance for the passenger group the passenger group size is multiplied by the passenger distance and by the overlap ratio; by multiplying the distance by the passenger group size the difference in passenger group sizes in taken into account. The result of this calculation is added to the previously declared time_slice_distance variable. The

other previously declared variable, time_slice_time, is also updated here, with the duration time of the overlap multiplied by the passenger group size being added to it. Once all of the passenger groups have been processed, the variables time_slice_distance and time_slice_time contain the total distance and time magnitudes for a given time-slice/trip-time-type pair.

With these values known, not only can the distance and time magnitudes be written to their respective, previously declared arrays, but the speed magnitude can also be written to its array, as speed is distance divided by time. Before each of these values is written to its array, a test is made to see if there was any activity during the time slice, or whether the magnitude of the respective values was zero. The time magnitude is used to gauge activity for the speed magnitude, an important test as, with speed being distance divided by time, a divide by zero error is possible. If any of the magnitudes are zero then a return character[198] is written. If any of the magnitudes are non-zero then the values are written out using convenient units, specifically kilometres for the distance, minutes for the time, and kilometres per hour for the speed.

Once all of the time-slices/trip-time-type pairs have been cycled through, the previously declared strings contain distance, time and speed data for all of the time periods and for each of the three trip-time types, and so can be written out to file for later analysis. For the distance, only the transit time type is written out. Although the distance values will move around the time slices somewhat for the different trip-time types, the total distance for all time slice will be the same, and the transit trip-

198 Writing a return character, also known as a carriage return character, results in a blank line appearing in the output file.

time types seems the most applicable as it shows the passenger distance on the line at the time when the passengers are actually travelling on the line. For the time, all three trip-time types are written out, as the wait, transit and journey times are all different and have quite distinct meanings. Finally, transit and journey speed are written out, with wait speed skipped as, although it has a clear definition, it does not have a clear meaning.

As well as compiling data separated out on the basis of time slice and trip-time-type, the AnalysePaxs function also calculates some metrics for the entire day's run. This task is begun by declaring and zeroing five variables for storing overall distance, journey time, transit time, wait time and passenger numbers.[199] As with compiling the time slice based data, the all-day data is accumulated by cycling over all of the passenger groups, and checking to see if the passenger groups are travelling in the desired direction. If so then the five variables have the individual passenger group distance, time and group size data added to them. With the exception of the passenger number variable itself, the values for the rest of the variables are multiplied by the passenger group size. When all of the passengers have been processed, six metrics are compiled from the five variables, namely average wait time, average transit time, average journey time, average transit speed, average journey speed and total distance. Average wait, transit and journey times are calculated by dividing the overall wait or transit or journey time by the overall passenger numbers, and is reported in whole seconds. Average transit and journey speeds are calculated by dividing the overall transit or

199 These five variable are respectively called overall_distance, overall_journey_time, overall_transit_time, overall_wait_time and overall_pax_size.

journey distance by the overall passenger numbers, and is reported in kilometres per hour. Total distance is directly available and is reported in kilometres. Once calculated, all of the metrics are written out to a file for later use, which completes the compiling of passenger data.

5.13 Analysing the Runs

The task of analysing the runs, menu option “i” of the Matilda software, which can been seen at the bottom left of the Figure 4.7 data-flow diagram, begins with the calculation of run drain levels for the ultra-efficient timetable, the number of buses out in service at any given time. The method used is essentially identical to the method used to produce the equivalent data for the May 2005 TransMilenio S.A. timetable. This method was discussed in detail at the end of section 5.4, “Converting the TransMilenio Run Files”, and so will not be covered again here.

In a similar vein, the task of compiling the run analysis data is very similar to that of compiling the passenger data, and so uses a structurally very similar body of code to compile the data. Consequently discussion of how the run analysis data has been compiled will be brief and only highlight the differences between compiling the run data and compiling the passenger group data, rather than re-detailing the code described in the previous section.

The most significant difference between compiling the run data and compiling the passenger group data, is that there is no concept of trip-time type used when compiling the run data. Passengers have a transit time, which occurs on the line, a wait time, that occurs at the stop, and a

journey time that is the sum of these two times. For runs, there is only one concept of time that has meaning, the transit time on the line.

Another significant difference between compiling the passenger and run data is that for a passenger there is only one notion of distance, the distance the passenger travels, whereas with runs there are three significant distances, the run distance, the passenger distance and the combined seated and standing place distance. In a paper by Siemens Transport International called *The Potential of Fully Automated Flexible Metro Systems,* the combined seated and standing capacity of a transit vehicle is referred to simply as places; a nomenclature this thesis will follow.[200] Although the place distance for the runs is, by definition, the run distance multiplied by the bus capacity, there is significant utility in reporting place distance as it allows, through comparison to the passenger distance data, occupancy calculations to be readily made. The three types of run distance are allowed for by, in the place where a single passenger distance variable was declared, declaring three variables, one for each of the distance types.

There are no further substantive differences between the manner in which the passenger data is compiled and the run data is compiled. As with the passenger data compilation, for the run data both time slice data and overall data is compiled, and all compiled data is written out for future use.

200 Siemens Transport International (1998), *The Potential of Fully Automated Flexible Metro Systems*, Siemens Transport International, Paris, pp 5-6

5.14 Conclusion

In conclusion, this chapter has detailed the workings of the Matilda software. Three main areas have been covered, namely preparing the data for processing, processing the data, and preparing the results for analysis. Preparing the data for processing involved the Matlida menu options “a” through “c”, namely “calculate line ratio”, “convert transmilenio run files to matilda run files” and “convert transmilenio load files to matilda load files.” Processing the data involved Matilda menu options “d” and “e”, namely “make run groups from loads” and “deadhead runs.” Preparing the results from processing the data for analysis involved Matilda menu options “f” through “i”, namely “make pax groups from run groups”, “make pax groups from loads”, “produce pax group analysis data” and “produce run group analysis data.” While explaining the operation of the code called by the selection of these menu options, not only has the fine detail of the methods used to produce the output data been covered, but also any approximations or assumptions used have been highlighted.

Showing the operation of the code down to a fine level of detail, and highlighting approximations and assumptions that were used has many advantages. A key advantage is that the correctness of the approach can be judged, for example, have computational errors been made? Another advantage is, in a case where a judgement has been made to calculate a certain aspect in a certain manner, the exact method of calculation can be seen. This allows readers who believe the judgement to calculate a certain aspect in a certain manner to have been a flawed judgement some realistic opportunity to gauge the impact of this ‘flawed judgement’ on

the output data. As such, a reader who is interested in the general thrust of this work, but disagrees with one or more of the judgement calls, can still gain some value from this work.

In the next chapter, the output data produced by the code described in this chapter, will be presented and analysed. By exposing the workings of the ‘computational machine’ that has produced this output data, the reader knows of the precise nature of the data to be presented, and therefore is in a position to be able to judge for themselves its soundness.

Chapter 6

Timetable Development Results and Analysis

6.1 Introduction

The primary task of this thesis has been to develop a software package to test the proposition that significant efficiency gains are achievable by timetabling Bus Rapid Transit in an ultra-efficient manner. In the previous chapter the operation of a prototype ultra-efficient timetabling software package, called Matilda, was discussed in detail. In chapter three, data suitable for processing by this software package, data for the TransMilenio's Américas Line, was prepared for processing by isolating it from the rest of the TransMilenio BRT system. This chapter will present and analyse the output data that resulted from using the Matilda software package to process the Américas Line data.

As has been discussed in various earlier sections, the Matilda software package implements one method of timetabling BRT in an ultra-efficient manner; no claim is made that the approach being used is the best or only method. Indeed, so as to both acknowledge and reinforce the fact that better methods might be developed, the Matilda software package has been repeatedly referred to as a prototype software package. That said, compared to the method used to develop the May 2005 TransMilenio S.A. timetable, the method employed by the Matilda software package is of quite a different character. The May 2005 TransMilenio S.A. timetable was developed by manually assessing one stopping pattern after another, with the assistance of a spreadsheet, searching for express service stopping patterns to be run over periods of

at least several hours.[201] The ultra-efficient timetabling method used by the Matilda software package automatically searches through and assesses up to approximately 200 thousand express service stopping patterns for each 15 minute time slice, both inbound and outbound. The results of that search, the performance of the best sets of express services that the Matilda software package found for the Américas Line, is presented in this chapter. Where relevant, the performance of the May 2005 TransMilenio S.A. efficient timetable is also presented for comparison purposes. A complete listing of the Américas Line ultra-efficient timetable generated by the Matilda software package is available in appendix two.

In the previous chapter the data transformation methods used to process the bus run and passenger data into a form suitable for presentation were discussed. These two datasets, the bus dataset and the passenger dataset, will be presented and analysed in this chapter. Buses are a resource that is deployed to provide a service, the service of mobility. As such, buses will be analysed in this chapter in terms that reflect the quantity of the resource that is being 'consumed.' Throughout this chapter buses will be analysed on the basis such as, for example, the number of buses required, the distance travelled by the buses or the time that the buses spend servicing their allocated runs. By comparison, passengers are 'consumers' of the service of mobility. As such, passengers will be analysed in this chapter primarily in terms that measure the quality of the mobility service they receive. Passengers will be analysed in terms, for example, of how long they had to wait at a stop

201 From notes taken at a meeting between the author and Alejandro Niño of the TransMilenio S.A. Planning Department on the 7th of June 2006.

before their bus service arrived, or how quickly they were transported to their destination stop.

This chapter will first present and analyse data on the functioning of the metric used to select express service stopping patterns. Next, the bus data produced by the Matilda software will be presented and analysed, followed by a similar presentation and analysis of the passenger data. Where possible comparisons will be drawn between the ultra-efficient timetable produced by the Matilda software, and the existing May 2005 efficient timetable. Where appropriate, the differences between the efficient and ultra-efficient timetables will be expressed in dollar terms. Some comparisons will be drawn to non-BRT transport modes, where instructive. After the bus and passenger data has been presented separately, the relationship between the bus and passenger data for ultra-efficient timetabling will be examined, and this relationship compared to the relationship between the bus and passenger data for the existing May 2005 efficient timetable. With all of the ultra-efficient timetable data presented and analysed, the question of whether the ultra-efficient timetable data reliably represents what the actual performance of such a timetable would be, will be addressed. Finally, issues relating to the operationalisation of an ultra-efficient timetable will discussed.

6.2 Combinations Searched

As discussed in the previous chapter, the basic method used to produce an ultra-efficient timetable has been to search through up to approximately 65 thousand stopping pattern combinations for each express service in operation. Each of these stopping patterns has, in turn,

been judged using a metric, and if the combination under assessment was better than the current best combination, the combination being assessed became the new best combination. The metric operated by adding up the number of intermediate, or middle, stops that would be stopped at by a given express service and the remaining all-stop buses that would be needed, and favouring a lower number of stops. Data showing the operation of this metric, on a small selection of the many combinatorial searches that were conducted, is shown in Figure 6.1 below.

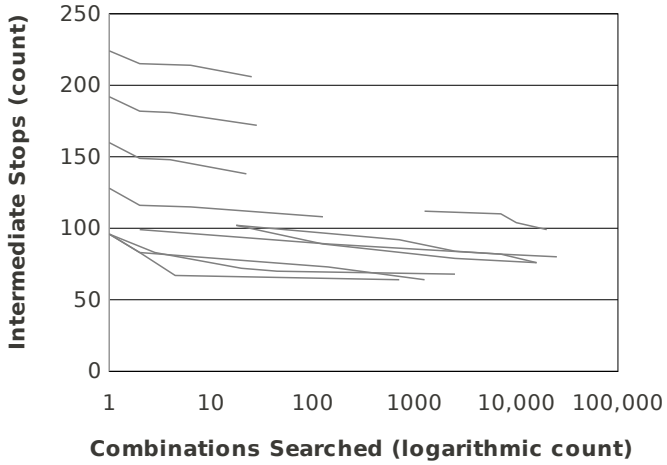


Figure 6.1: Intermediate Stops by Combinations Searched

Of the many, many combinatorial searches performed, eleven are shown in Figure 6.1. Only those searches that were assessing express services going to the far stop, stop 17, were included, with near stop, stop 13, searches discarded. The near stop searches, where approximately four thousand stopping patterns are assessed, were discarded so that all the data presented would be from searches of the same ‘duration’, namely the approximately 65 thousand stopping patterns of the far stop searches. Also, all searches with less than four

improvements in the metric were also discarded, so as to reduce visual clutter on the graph. An assessment was made of the data and, except where has just been indicated otherwise, the data used was representative of the total data-set. A logarithmic scale was used on the X axis of Figure 6.1, as a significant number of improvements in the metric occur during the first ten combinations, a fact that would have been obscured if a linear scale had been used.

The combinatorial searches in Figure 6.1 show, on the whole, gradual improvements occurring, with lower, and therefore better, metric scores slowly being achieved as more combinations are searched. As the total number of runs being targeted varies significantly between combinatorial searches, the improvements in the metric scores are accordingly in different intermediate stop ranges, with a group of combinatorial searches at around the 100 intermediate stops level and then others up to above 200 intermediate stops. The combination ranges over which the improvements in the metric scores occur also vary significantly, with some combinatorial searches improving immediately whereas others do not improve until a substantial number of combinations have been assessed. In one case, no improvement occurs until over one thousand combinations have been assessed.

Although both the intermediate stop ranges and the combination ranges of the combinatorial searches shown in Figure 6.1 vary significantly, the curves shown are very similar in character, with gradual, rather than abrupt, improvements occurring. At the end of each of these sequences of improvements, a final ‘best’ express pattern is

selected for use. The remainder of this chapter reports on how well these final selected express patterns perform.

6.3 Bus Drain

At any given time a given BRT line will have a certain number of buses on the line servicing passenger load, a level that has been referred to in this thesis as the bus drain. This metric is important because the peak bus drain level determines the number of buses that need to be acquired, and buses are a significant capital cost. This section will present the bus drain for the ultra-efficient timetable and, to the degree that it is reliable, compare it to the bus drain for the efficient timetable. To ensure consistency between the efficient and ultra-efficient timetable data presented in this section, and following sections, all bus metrics for the ultra-efficient timetable have been calculated using total bus kilometres, revenue plus deadheading kilometres, rather than just revenue kilometres. The ultra-efficient timetable makes very extensive use of deadheading, whereas the efficient timetable for the Américas Line, at most, makes negligible use of deadheading, so to do otherwise would markedly skew the results in favour of the ultra-efficient timetable. In Figure 6.2 below, the bus drain for the ultra-efficient and efficient timetables is compared across the day.

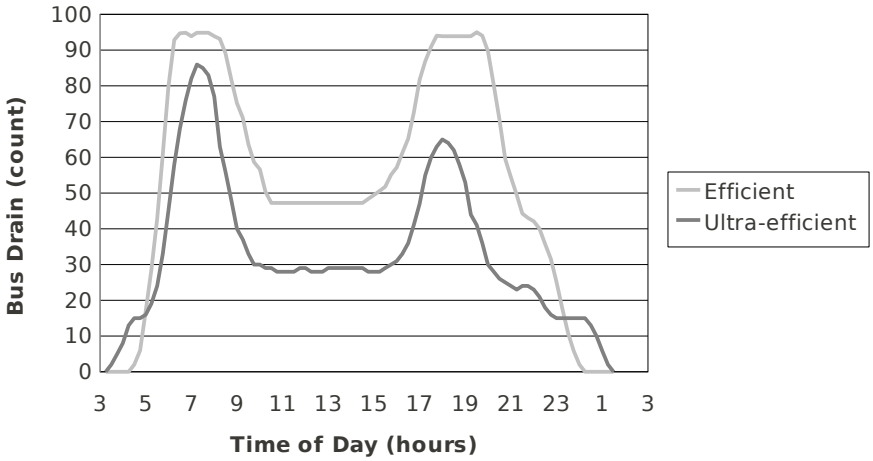


Figure 6.2: Comparative Bus Drain by Time of Day

Timetabling Method	Efficient	Ultra-efficient
Maximum Bus Drain (count)	95	86

Table 6.1: Comparative Maximum 24-Hour Bus Drain

One complexity in constructing the above graph is that the May 2005 efficient timetable operates buses out of a number of depots whereas the ultra-efficient timetable uses only one depot. The maximum number of buses for the May 2005 timetable could considered to be 92, 95 or 100 buses, depending on how the buses are counted. A maximum bus drain of 92 buses may not actually be physically achievable, as it would involve swapping buses between depots and it is not clear that the necessary time is available. The maximum bus drain of 95 buses does appear to be physically achievable and so, although it is not clear from the received data that this is how the buses are actually timetabled, this number has been used to generate Figure 6.2.

The May 2005 efficient timetable peaks at a maximum bus drain of 95 buses, whereas the ultra-efficient timetable peaks at a maximum bus drain of 86 buses, a reduction of approximately 9%. With the articulated

buses used on the TransMilenio system costing approximately US$200 thousand each,[202] in financial terms this represents a capital cost saving of approximately US$1.8 million for the Américas Line. Dividing this saving between the 186,000 maximum boarding level for the Américas Line,[203] gives a capital cost saving of approximately US$10 per boarding serviced. If the reduction in buses required is scaled to the TransMilenio system as a whole, using the boarding figure for the whole system of 1,450,000, as per the 2007 *Bus Rapid Transit Planning Guide*,[204] then the total capital cost saving would be approximately US$14 million.

The difference in the maximum bus drain is normally greater than 9%, with the difference between the efficient and ultra-efficient timetable being most marked in the afternoon. Figure 6.3 below takes the bus drain data from Figure 6.2 and shows it as a ratio between the bus drain of the efficient and ultra-efficient timetables.

202 *The Transmilenio: An Excellent Cost-effective Solution*, The Institut Veolia Environment Website, Available: http://www.institut.veolia.org/en/cahiers/urban-transport/bogota/transmilenio.aspx (Accessed 16 June 2009)

203 Cain, A. (Principal Investigator) (2006), *Applicability of Bogotá's TransMilenio BRT System to the United States - Final Report May 2006*, Federal Transport Administration, Department of Transportation, United States of America, Washington, D.C., U.S.A., p. 20

204 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, p. 767

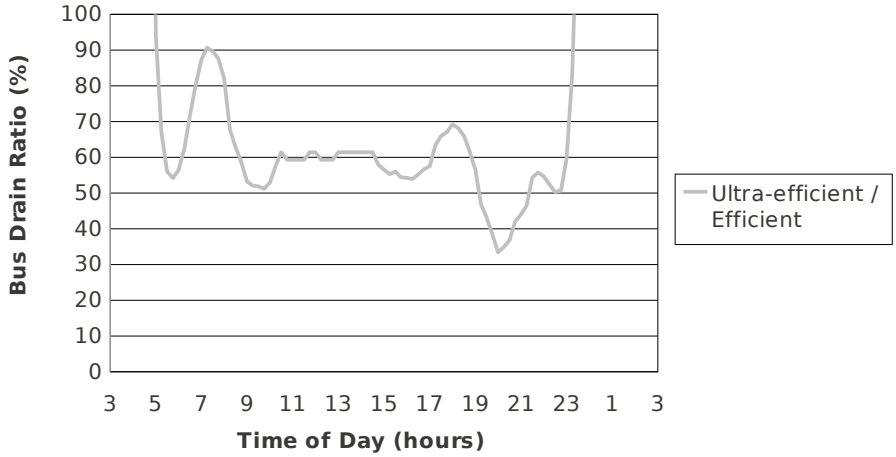


Figure 6.3: Ultra-efficient Timetable/Efficient Timetable Bus Drain Ratio by Time of Day

Timetabling Methods	**Ultra-efficient/Efficient**
Average Ratio (%)	63%

Table 6.2: Average Ultra-efficient Timetable/Efficient Timetable 24-Hour Bus Drain Ratio

The above graph would strongly indicate that the efficient timetable's bus services, which vary in frequency but not stopping pattern, have had their stopping patterns specifically tuned to the morning peak. This would be a sensible measure to meet the objective of minimising the maximum number of buses required to run the line, and therefore minimising this aspect of the line's capital cost. Outside of the morning peak, the difference between efficient and ultra-efficient timetables is marked. On average, the ultra-efficient timetable has only 63% of the buses in use compared to the efficient timetable.

6.4 Bus Distance

Another way of comparing the efficient and ultra-efficient timetables is to look at the comparative bus kilometres timetabled by each. This

metric is important because the magnitude of many of the costs associated with operating a BRT line, such as tyres and fuel, are determined by bus distance. Figure 6.4 below compares the total bus kilometres, revenue and deadheading, travelled by the efficient and ultra-efficient timetables.

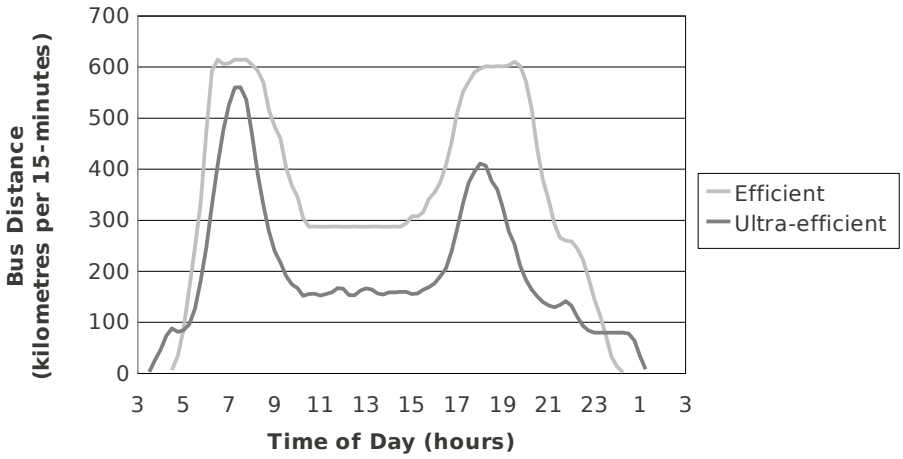


Figure 6.4: Comparative Total Bus Kilometres by Time of Day

Timetabling Method	Efficient	Ultra-efficient
Total Bus Distance (kilometres)	29,708	17,648

Table 6.3: Total Comparative 24-Hour Total Bus Kilometres

As would be expected, the shape of the above bus kilometres graph is roughly similar to Figure 6.2, the bus drain graph. In total, the ultra-efficient timetable travels approximately 40% less kilometres than the efficient timetable. A paper from 2005, *Study of Urban Public Transport Conditions in Bogotá, Colombia*, says that transparent data on TransMilenio BRT operating costs is not available, but that the best available data is the fee paid to the bus operators of US$1.92 per kilometre.[205] This figure is composed of allowances for operating and

205 *Study of Urban Public Transport Conditions in Bogotá, Colombia*, The Public-Private Infrastructure Advisory Facility Website, Available: http://www.ppiaf.org/

maintenance costs, such as diesel, tyres, lubrication oil, drivers, mechanics, administration personnel and expenses, as well as allowances for insurance, depreciation and return on capital.[206]

From Table 6.3 above, the ultra-efficient timetable services the Américas Line using 12,060 less bus kilometres per weekday than the efficient timetable. Multiplying this figure by an operating cost of US$1.92 per kilometre, gives a operating cost saving of approximately US$23 thousand per weekday, or a saving of approximately US$6.0 million over a year. If the reduction in bus kilometres is scaled to the TransMilenio system as a whole, using the same boarding ratio used to scale the bus drain data in the previous section, then the weekday operating cost saving for the TransMilenio system as a whole would be approximately US$181 thousand per weekday or US$47 million per year.

This section, in conjunction with the previous section has addressed the eighth and central research question of this thesis, namely whether a BRT timetable which varies its service frequencies and stopping patterns at the same pace that passenger load levels and patterns change can significantly outperform a normal BRT timetable. This question has been answered in the positive. In the previous section, section 6.3, “Bus Drain”, bus fleet size under the ultra-efficient timetable was shown to be 9% lower than for the existing timetable. In this section, bus kilometres travelled under the ultra-efficient timetable have been shown to be 40% lower than for the existing timetable. In the introduction, research question eight was rephrased as a thesis, with the thesis to be tested

documents/toolkits/UrbanBusToolkit/assets/CaseStudies/full_case/Bogata.doc (Accessed 16 June 2009), p. 75

206 ibid.

being: *whether a BRT timetable which varies its service frequencies and stopping patterns at the same pace that passenger load levels and patterns change, can significantly outperform a normal BRT timetable.* This thesis has been confirmed.

6.5 Bus Time

A final way of assessing efficient versus ultra-efficient timetables, on a purely vehicular basis, is to compare the time that buses spend on the line. This metric is important because, as each bus needs a driver, bus time is the primary determinant of driver time and therefore driver cost. Figure 6.5 below compares the total bus hours, revenue and deadheading, travelled by the efficient and ultra-efficient timetables.

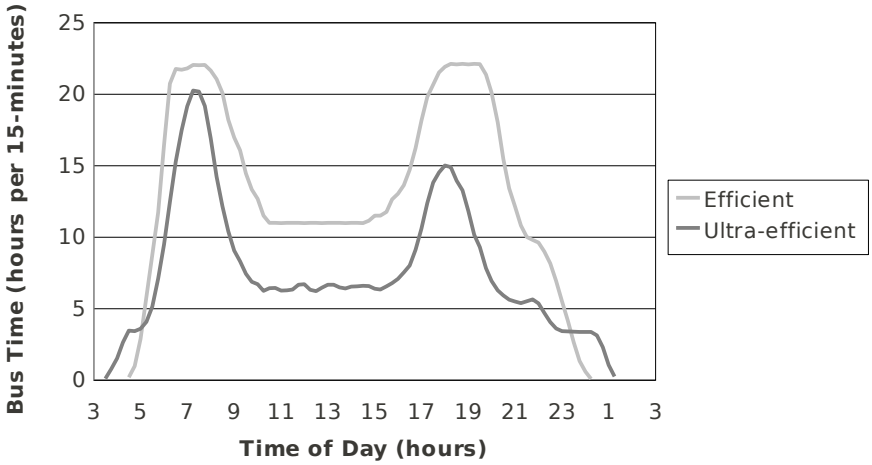


Figure 6.5: Comparative Total Bus Time by Time of Day

Timetabling Method	Efficient	Ultra-efficient
Total Bus Time (hours)	1,082	678

Table 6.4: Total Comparative 24-Hour Total Bus Time

In total, the ultra-efficient timetabling approach uses approximately 37% less bus time than the efficient timetabling approach. The financial impact of this time reduction was, in effect, assessed in the last section on bus distance, and so will not be considered further. The similarity of the 37% bus time reduction figure to the bus distance reduction figure of 40%, simply reinforces the operating cost saving assessments of the last section.

6.6 Bus Occupancy

The previous three sections have presented data that is purely based on the buses, namely bus drain, bus distance and bus time, whereas the next sections will present data purely based on the passengers. Uniquely, this section will present data, namely bus occupancy, that, although it is bus focused, draws from both the bus and passenger data. Detailed occupancy data for the May 2005 efficient timetable was not readily available and would have been difficult to construct, as calculating occupancy levels involves simultaneously handling bus and passenger data. As detailed occupancy data for the efficient timetable was not considered essential in making the argument of this thesis, such data was not constructed. Figure 6.6 below, presents the occupancy data, across the day, for the ultra-efficient timetabling method, split into inbound and outbound directions.

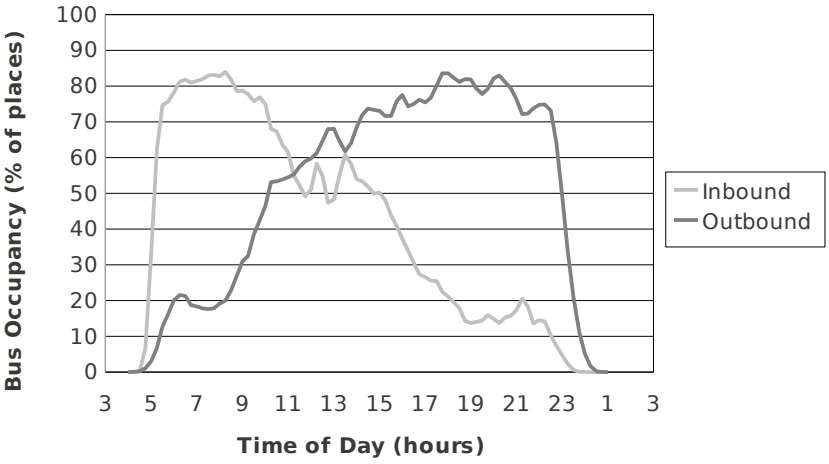


Figure 6.6: Ultra-efficient Timetable Inbound and Outbound Bus Occupancy by Time of Day

Direction	Inbound	Outbound
Average Bus Occupancy (% of places)[207]	49	51

Table 6.5: Average Ultra-efficient Timetable Inbound and Outbound 24-Hour Bus Occupancy

As would be expected, the ultra-efficient timetabling method achieves its highest bus occupancy levels during the morning and afternoon periods of high passenger load, in the direction of the high passenger load. In the opposite direction to the high passenger loads, passenger loads are comparatively low and this is reflected in the poor occupancy levels outbound in the morning and inbound in the afternoon, with extensive use being made of deadheading during these times. Figure 6.7 below, combines the inbound and outbound directions to present the occupancy data for both directions.

207 As specified in section 5.13, “Analysing the Runs”, the term “places” refers to the combined seated and standing capacity of the buses. In section 2.2, “What is Bus Rapid Transit?”, it was discussed that the TransMilenio S.A. rate their buses as having a total capacity of 160 passengers, which is the capacity figure that has been used to calculate occupancy in this section, and for other purposes throughout this thesis.

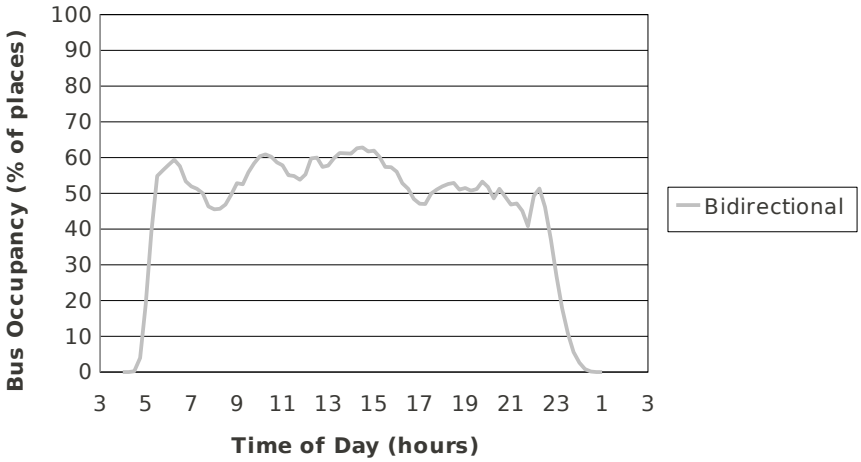


Figure 6.7: Ultra-efficient Timetable Bidirectional Bus Occupancy by Time of Day

Direction	**Bidirectional**
Average Bus Occupancy (%)	50

Table 6.6: Average Ultra-efficient Timetable Bidirectional 24-Hour Bus Occupancy

Overall bidirectional occupancy is relatively stable in the 45% to 60% range. High passenger load periods would, all else being equal, have higher occupancy levels, but these levels are being suppressed by how unbalanced the passenger load is during these times. Over time, development around the stops, a basic principle of transit orientated development, might be able to dampen down this level of imbalance by generating more contra-flow passenger demand.

A key determinant of whether such transit orientated development concepts can be effectively applied, is the distance between stops. In a paper entitled *Bursting the Bubble: Determining the Transit-Oriented Development's Walkable Limits*, it is noted that “transit-oriented developments (TODs) in the United States have been modelled almost exclusively with a half-mile radius as a reliable limit for pedestrian

walkability from and to a light rail station. New research has emerged to challenge this standard, with data indicating that transit users may be apt to walk greater distances than previously estimated."[208] Both the Américas Line, with an average stop spacing of 782 metres, and the TransMilenio system as a whole, with an average stop spacing of 500 metres,[209] are within the uncontroversial half-mile (805 metre) TOD walkability limit. Such development around stops may already be in progress, with a World Bank report noting that "it is interesting to notice some first signs of the impacts of TransMilenio on land use in Bogotá, with new shopping centres being constructed next to some busways."[210]

The current average occupancy on the Américas Line, calculated from the received efficient timetable data, is 29.6%, or approximately 40% lower than the ultra-efficient timetable's average occupancy level of 50%. This significant difference in occupancy is consistent with the significant reduction in bus distance and bus time by the ultra-efficient timetable, discussed in previous sections. When timetabling the same system, the only way to run buses less is to run them fuller.

Interestingly, in terms of occupancy, both the efficient and ultra-efficient timetables perform well compared to some other bus-based systems. For example, an article in the London Telegraph newspaper, reports that "averaged over the whole day [for the London bus system], there are just 14.5 passengers in each bus. Given that the average

208 Canepa, Brian (2007), *Bursting the Bubble: Determining the Transit-Oriented Development's Walkable Limits*, Transportation Research Record, Volume 1992

209 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, p. 767

210 *TransMilenio Busway-Based Mass Transit, Bogotá, Colombia*, The World Bank Website, Available: http://siteresources.worldbank.org/INTURBANTRANSPORT/Resources/Factsheet-TransMilenio.pdf (Accessed 17 January 2009), p. 14

capacity is 90, that's a load factor of only 16%."[211] By comparison, the TransMilenio S.A.'s efficient timetable for the Américas Line, in delivering an occupancy level of 29.6%, is doing nearly twice as well; a fact that may go a long way to explaining why the TransMilenio is able to run without an operating subsidy,[212] whereas the subsidy for the London bus system in 2004 was £600 million.[213]

6.7 Passenger Wait Time

This section switches the analysis of the ultra-efficient timetable data over to analysing the passenger data, beginning with how long passengers have to wait at their stops, under the ultra-efficient timetabling method. Figure 6.8 below presents the wait time for the passengers, across the day.

211 *Hop on, Conductor Ken is Taking Us All for a Ride*, The Telegraph Website, Available: http://www.telegraph.co.uk/finance/2880453/Hop-on,-Conductor-Ken-is-taking-us-all-for-a-ride.html (Accessed 24 January 2009)

212 Cain, A. (Principal Investigator) (2006), *Applicability of Bogotá's TransMilenio BRT System to the United States - Final Report May 2006*, Federal Transport Administration, Department of Transportation, United States of America, Washington, D.C., U.S.A., p. 21

213 *Hop on, Conductor Ken is Taking Us All for a Ride*, The Telegraph Website, Available: http://www.telegraph.co.uk/finance/2880453/Hop-on,-Conductor-Ken-is-taking-us-all-for-a-ride.html (Accessed 24 January 2009)

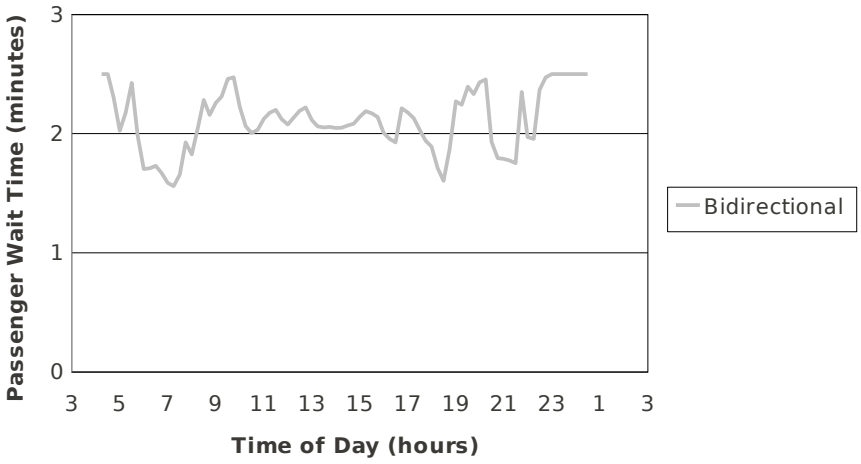


Figure 6.8: Ultra-efficient Timetable Passenger Wait Time by Time of Day

Timetabling Method	Ultra-efficient
Average Passenger Wait Time (minutes)	2:00

Table 6.7: Average Ultra-efficient Timetable 24-Hour Passenger Wait Time

Figure 6.8 shows a sharp upper-bound of exactly two and a half minutes for passenger wait time over the entirety of the day. In producing the ultra-efficient timetable a minimum bus frequency of three buses every 15 minutes was used in the ultra-efficient timetable development, to match, as closely as possible, the existing efficient timetable's minimum bus frequency. This minimum bus frequency was both evident in the received data, and re-affirmed by the *Bus Rapid Transit Planning Guide's* reporting of the Transmilenio's average non-peak headway as being five minutes.[214] One bus every five minutes caps the maximum wait time at two and a half minutes. The average, as opposed to the maximum wait time, came out only slightly less than this two and a half minute cap, at exactly two minutes.

214 Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York, p. 767

In a similar fashion to the May 2005 efficient timetable, the ultra-efficient timetable runs one all-stop bus service and a variable number of express services, at any given time. As passenger load increases, more express services are established, and so, in Figure 6.8, wait times do not drop as much as might be expected during high-load times. Rather than existing bus services being run more frequently as passenger load increases, the number of express services in use is expanded. For example, consider a case where passenger load was sufficient to justify running three all-stop buses every fifteen minutes, giving one bus every five minutes and so an average wait time of two and a half minutes. If passenger load were to double, the run frequency of the all-stop service could also be doubled, with the result being a halving of the average wait time. By contrast, what actually occurs, under ultra-efficient timetabling, is that an express service is established alongside the all-stop service, giving two separate services both running three buses every 15 minutes, and consequently an average passenger wait time unchanged at two and a half minutes. Such neat transitions, with passenger load precisely doubling, do not in practice occur, and so some fluctuation in wait time does occur across the day, as observable in Figure 6.8. Nevertheless, the favouring of an expansion in services over an increase in the frequency of existing services, explains why wait times do not drop greatly below their two and a half minute upper-bound.

The one clear case, in Figure 6.8 above, where there is a sustained drop in wait times, significantly below the two and a half minute upper-bound, is during the morning peak. This sustained drop in wait times is due to the total allowable services being limited to four, one all-stop

service and three express services. In section 5.6, “Making the Runs: Data Preparation and Pre-calculations”, it was discussed that the peak passenger load on the Américas Line was sufficient to justify up to eight services, one all-stop and seven express services, but that a cap of four total services had been put in place. The purpose of this cap was to mirror the existing efficient timetable’s use of only four services and thereby take into consideration, at least as far as reasonably possible, limited bus-bay availability at the stops. During the morning peak, passenger load is sufficient to support the establishment of five to eight total services, but, with this option having been explicitly disallowed, the load is instead serviced by increasing the frequency of services. During the morning peak, services are regularly running at a frequency of between four and seven buses every 15 minutes, thus giving lower wait times. To a lessor degree, this effect of ‘running out’ of allowed services to establish, and therefore increasing service frequency instead, can also be seen during the afternoon peak.

6.8 Passenger Speed

This section presents the speed at which passengers travel under the ultra-efficient timetable, and is the final section whose primary focus is on presenting and analysing the ultra-efficient timetable output data. Further sections will either be directly comparing the efficient and ultra-efficient timetables, or discussing more general matters, such as the issues associated with the operationalisation of an ultra-efficient timetable.

Passenger transit and passenger journey speeds are presented in this section. Transit speed is a classical transport metric, being the speed at which a passenger travels, from the moment they board a transit vehicle to the moment they alight. Transit speed does not take into account the time that a passenger spends waiting at their stop for a transit vehicle to arrive. By contrast, journey speed does take into account the time that a passenger spends waiting at their stop, and reports the speed at which a passenger travels, with the time being measured from the moment they enter their departure stop to the moment they alight their transit vehicle. Figure 6.9 below, presents the transit and journey speed data for ultra-efficient timetabling, across the day.

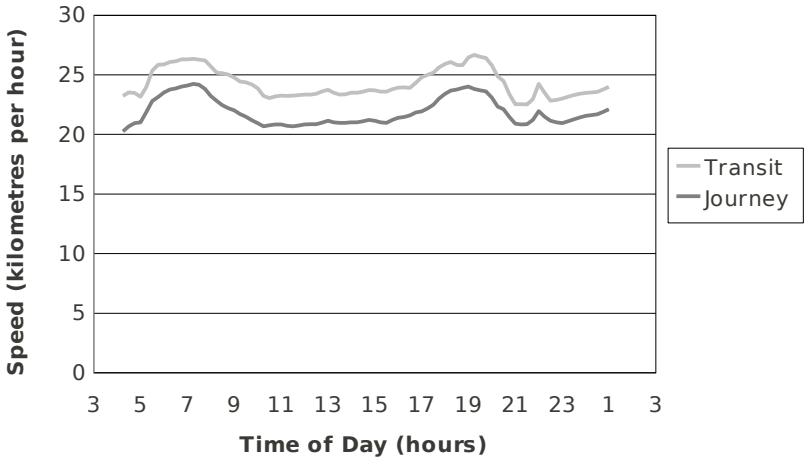


Figure 6.9: Ultra-efficient Timetable Passenger Transit and Journey Speed by Time of Day

Speed Type	Transit	Journey
Average Passenger Speed (km/h)	24.8	22.4

Table 6.8: Average Ultra-efficient Timetable 24-Hour Passenger Transit and Journey Speed

Both the transit and journey speeds in Figure 6.9 stay in a fairly tight range, with speed improvements of no more than approximately five

km/h during the morning and afternoon peaks. The reason for the speed improvements during these peak times is that more express services are being run, due to the higher loads, and so more passengers are travelling on express buses as compared to all-stop buses. The fact that the transit and journey speed curves are separated by almost exactly the same magnitude across the day, reflects the comparatively stable wait times that were shown previously in Figure 6.8.

The ultra-efficient timetable's average transit speed of 24.8 km/h is slightly lower than the efficient timetable's average transit speed of approximately 26 km/h.[215] This difference would be defrayed somewhat by the fact that the ultra-efficient timetable uses a no passenger transfer approach, thus eliminating the time passengers wait while transferring between services. Improving passenger travel speeds was not an objective of the software package developed for this thesis, and so having similar passenger speeds for the efficient and ultra-efficient timetables is not a surprise outcome.

The development of a software package aimed at improving efficiency has, though, provided an appropriate modelling tool with which to review the seemingly widely held view that BRT is a mid-speed transit technology. Professor Vuchic most clearly reflects this assessment of BRT as a mid-speed transit technology, with his classification of BRT as a semi-rapid form of transit.[216] It is a concern that BRT may have been classified as a mid-speed technology due to insufficient consideration

215 *Bogotá, What's Next?: by Dario Hidalgo* (2006), EMBARQ: The World Resources Institute Centre for Sustainable Transport Website, Available: http://www.embarq.wri.org/documentupload/WRI%20WB%20Jan%202006%20Dario%20Hidalgo.pdf (Accessed 17 February 2009), p. 6

216 Vuchic, V. R. (2007), *Urban Transit Systems and Technology*, John Wiley & Sons, New Jersey, p. 76

having been given to the fact that a technology, and how that technology is deployed, are not the same thing; the technology of BRT is not, for example, the TransMilenio BRT system. Assessments about the technology of BRT appear to have been made on the basis of general BRT deployments, which are, in fact, assessments about *BRT technology and how it has been deployed*, not about BRT itself. It may well be clear that BRT, as it has been deployed, is a mid-speed technology, but the only way of actually assessing BRT technology itself is via a like-by-like case. For example, how would BRT technology perform if it were assessed compared to a typical metro installation, such as the London Underground.

The London Underground has ten lines, 270 stops, and in total 402 kilometres of lines,[217] which gives an average stop spacing of 1,546 metres, compared to the average stop spacing on the Américas Line of 782 metres. At an average stop-spacing of 782 metres, the ultra-efficient timetable delivers a transit speed of 24.8 km/h, as shown in Table 6.8. Change this single variable, set the average stop spacing on the Américas Line to the 1,546 metre average stop spacing of the London Underground, and when the Matilda software is re-run it reports an average transit speed of 28.6 km/h. The average transit speed for the London Underground is 33 km/h.[218]

Passage along the Américas Line is, though, greatly impeded by 16 combined car/pedestrian traffic lights and 8 pedestrian only lights, along its 13.3 kilometre length.[219] With the scaling up of the line, to yield a stop

217 *Key Facts*, Transport for London Website, Available: http://www.tfl.gov.uk/corporate/modesoftransport/londonunderground/1608.aspx# (Accessed 19 February 2009)

218 ibid.

219 Count taken by the author during the June 2006 site visit.

spacing equivalent to that of the London Underground, the number of traffic and pedestrian lights was also scaled up, yielding approximately 32 combined car/pedestrian traffic lights and 16 pedestrian only lights along a 26.3 kilometre line. Removing the combined car/pedestrian traffic lights, via over or under passes, would be a significant expense, but this is not the case for the pedestrian only lights; light-weight steel pedestrian over-passes are a common feature on the TransMilenio system. If the interruption level along the Américas Line is reduced by one-third — the ratio of pedestrian only lights to total lights — when the Matilda software is re-run, still using the London Underground's average stop spacing, the software reports an average transit speed of 32.7 km/h.

So with only modest infra-structure changes, a version of the Américas Line, scaled to match the stop spacing of the London Underground, has BRT technology delivering an equivalent transit speed to Rail Rapid Technology (RRT). Indicative results only, but in terms of actually comparing BRT and RRT *technology*, no more conclusive results are available in the literature. Sophisticated discussions on the transit speed of BRT *deployments*, and how such speeds compares to RRT *deployments* are plentiful, but, as previously observed, a deployment is not its deployment technology.

Although it has not been an objective of the research presented in this thesis to improve BRT speed outcomes, the software tool developed to improve the efficiency outcomes of BRT systems can also be used to report on BRT transit speed, under various scenarios. Preliminary investigations into the transit speed performance of BRT, cast doubt on

the view that Bus Rapid Transit technology is slower than Rail Rapid Transit technology.

6.9 Comparing the Bus Run and Passenger Load Data

In section 3.9, “Comparing the Bus Run and Passenger Load Data” the bus runs for the May 2005 efficient timetable were compared to the passenger load those runs were servicing. The graph comparing these two factors, from section 3.9, is reproduced here.

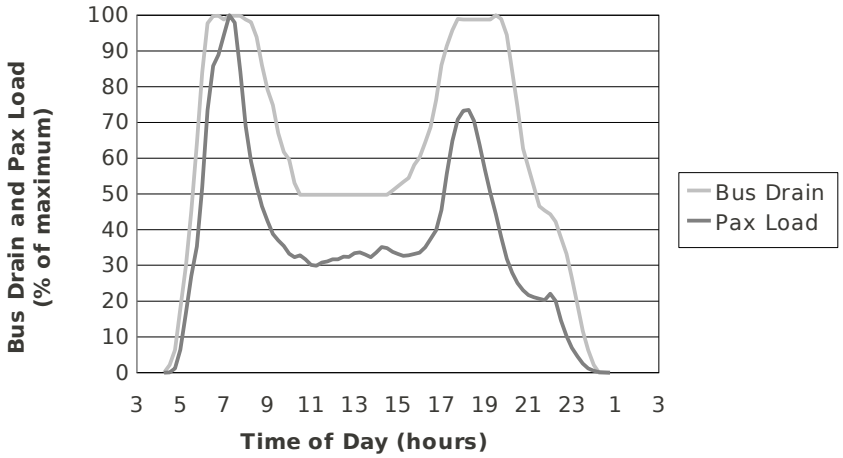


Figure 6.10: Efficient Timetable Bus Drain and Pax Load Compared by Time of Day

As previously discussed in section 3.9, although there is a clear correlation between passenger load levels and the bus drain in the above graph, the correlation for the efficient timetabling method is comparatively loose. Figure 6.11 below, is the equivalent graph for the ultra-efficient timetabling method.

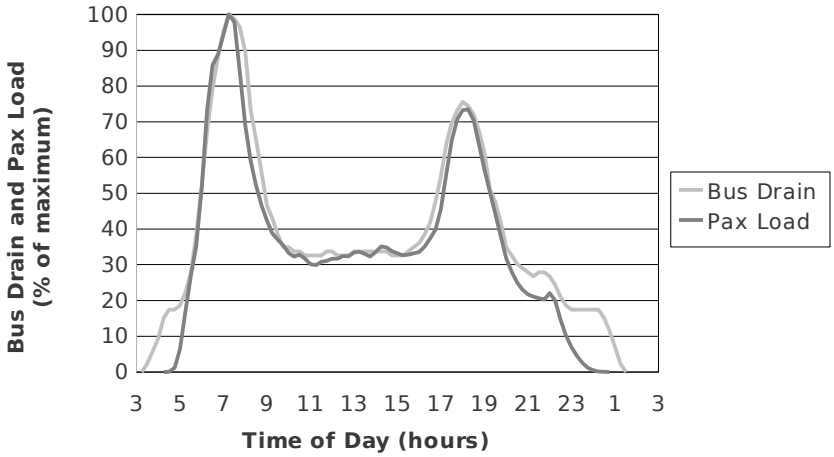


Figure 6.11: Ultra-efficient Timetable Bus Drain and Pax Load Compared by Time of Day

In Figure 6.11 a very tight correlation between bus and passenger levels can be seen. Indeed the difference between Figure 6.10, the efficient timetabling graph, and Figure 6.11, the ultra-efficient graph is quite striking. For much of the day, the bus drain curve for the ultra-efficient graph appears as though it has been draped over the passenger load curve. Another way of presenting the data in Figure 6.11 above, is to show the difference between the two data-sets. This difference can be viewed as a tracking error; a measure of how precisely the timetabling method is matching bus levels to passenger load.

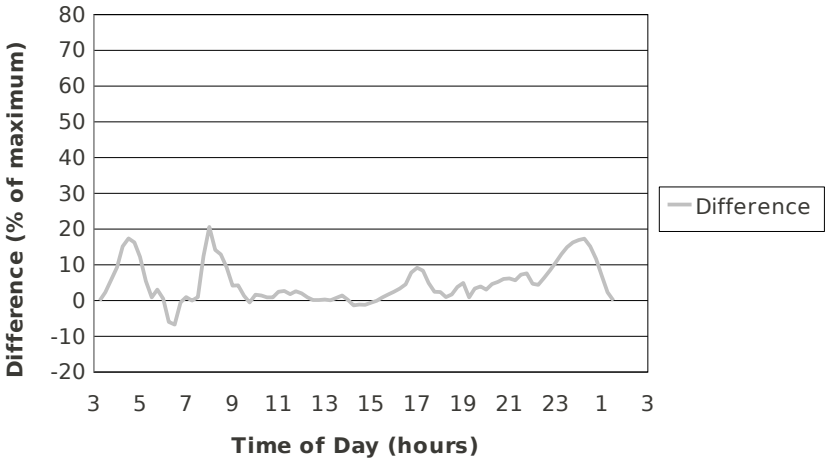


Figure 6.12: Ultra-efficient Timetable Tracking Error by Time of Day

Figure 6.12 shows the relatively constant relationship between bus drain and passenger load, for the ultra-efficient timetabling method. There are three time periods in Figure 6.12 where there is a significant tracking error; during the morning peak, during the afternoon peak and trailing off into the evening. During the morning and afternoon peaks, tracking-error spikes occur both as load is rapidly increasing and as it is rapidly decreasing; this is most clearly visible during the morning peak. The reason for these tracking-error spikes is that bus capacity can only be delivered in units of approximately one hour, the time needed to run a bus from the outermost stop to the innermost stop and back. When load changes sharply *within* an hour, as it does ramping up and down from the morning and afternoon peaks, precise tracking becomes unachievable. The tracking error trailing off into the evening is caused by a far simpler reason; the passenger load trailing off into the evening is low, and the minimum service guarantee is causing three buses to be run every fifteen minutes, even though the load does not justify that service frequency. Figure 6.13 below presents the equivalent tracking error graph for the

efficient timetabling method, with the ultra-efficient tracking error curve also shown for comparison purposes.

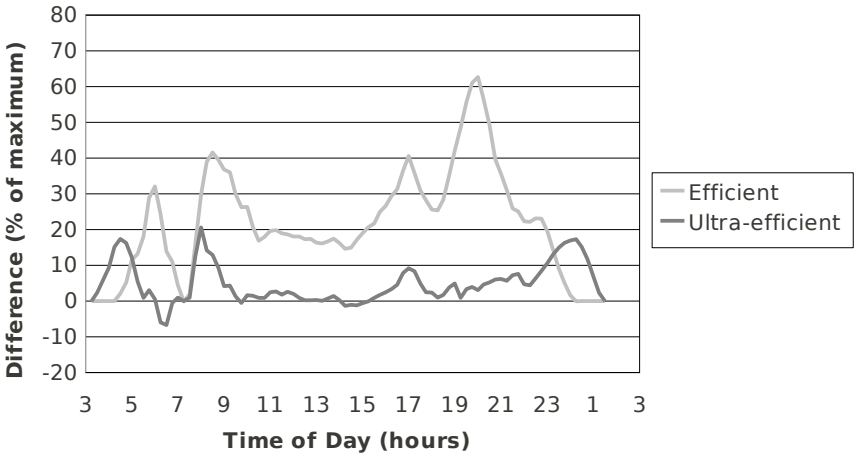


Figure 6.13: Efficient and Ultra-efficient Timetable Tracking Error by Time of Day

In Figure 6.13, it can be seen that the tracking error for the efficient timetabling method is, across the bulk of the day, far higher than for the ultra-efficient timetabling method. To show this difference in tracking error as a single value, the two curves above can be melded into a single curve. This has been done in Figure 6.14 below, by taking the absolute value of the tracking error, and then subtracting the ultra-efficient tracking error from the efficient tracking error. This calculation produces positive values when the ultra-efficient method is tracking passenger load more tightly than the efficient method, and negative values when the efficient method is tracking passenger load more tightly than the ultra-efficient method.

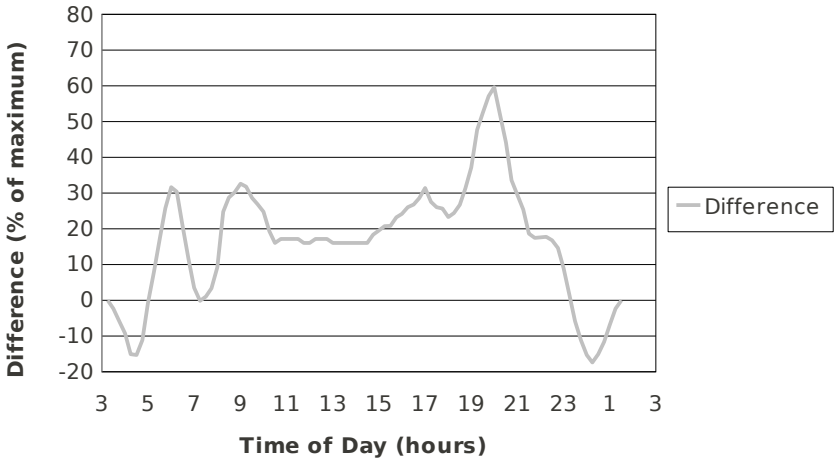


Figure 6.14: Tracking Error Difference by Time of Day

In Figure 6.14 above, it can be seen that, for the bulk of the day, the ultra-efficient timetabling method tracks passenger load far more precisely than the efficient timetabling method. This is a not unexpected result, given that the key characteristic of ultra-efficient timetabling, discussed in section 4.2, is the rapid tracking of passenger load.

6.10 A Pseudo-live Effect?

Before moving on to consider issues relating to the operationalisation of an ultra-efficient timetable, this section will examine the question of whether the, apparently very promising, results presented in this chapter can be relied upon. Specifically, the ultra-efficient timetable developed here has only been drawn from one set of data, and so a timetable has, in effect, been tailored for a particular passenger load data-set and then judged on the basis of its proficiency in servicing that same data-set; a situation that might be referred to as a pseudo-live effect. The ultra-efficient timetable is not actually live, it is just a very complex static timetable, but it has been constructed with effective perfect fore-

knowledge of the passenger loads it will be required to service. This was not the case with the efficient timetable, where the timetable was constructed by the TransMilenio S.A. using a projected data-set, as opposed to being specifically tailored to the received May 2005 data-set.

It is likely that some of the performance benefits reported in this chapter stem from the pseudo-live timetabling of the ultra-efficient timetable. That said, comparing the shapes of the efficient and ultra-efficient bus drain curves, in the absence of the May 2005 passenger load curve, would suggest that there is far more going on than just a pseudo-live effect. The shapes of the efficient and ultra-efficient bus drain curves were shown in Figure 6.2, and are reproduced below.

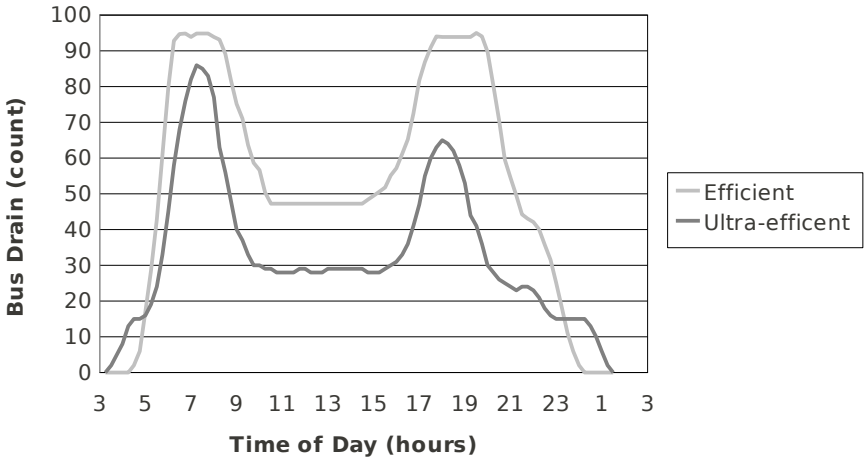


Figure 6.15: Efficient and Ultra-efficient Timetable Bus Drain Curves by Time of Day

As was noted in section 3.5, “The ‘Refined’ Bus Runs Data”, the efficient timetable bus drain curve is both symmetrical and also has angular transitions between lengthy service periods where the bus levels are held completely stable. This is not how passenger load curves look; people are simply not so regular. Figure 6.16 below is a graph of the

passenger load from a completely unrelated transit line, the Fremantle Railway Line in Perth,[220] with the TransMilenio's Américas Line passenger load data scaled and then overlaid for comparison.

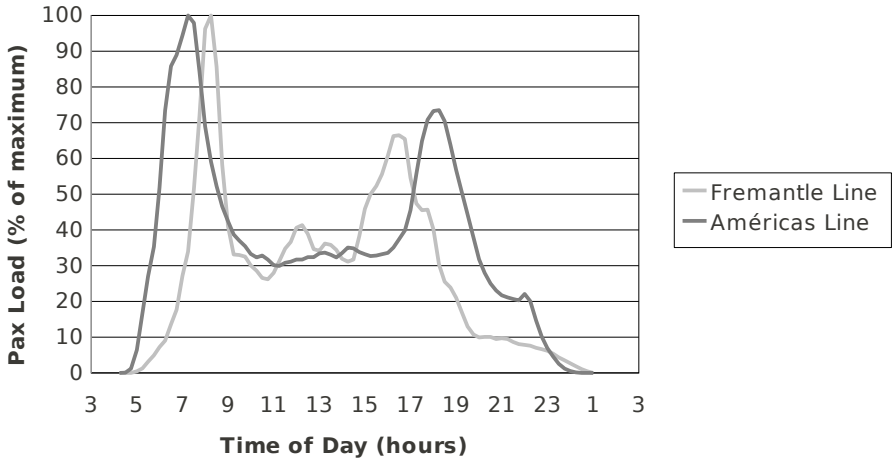


Figure 6.16: Passenger Load from Two Unrelated Systems by Time of Day

Although the above passenger load curves differ, coming as they do from two totally unrelated transit lines, they are of a very similar nature. Neither of the two passenger load curves in Figure 6.16 are symmetrical, neither exhibits angular transitions and neither has lengthy flat zones. The ultra-efficient timetable bus drain curve shown in Figure 6.15, is of the nature of the above passenger curves, and is so by design, not accident. The nature of the efficient timetable bus drain curve is not the same as the above passenger load curves, which, with apologies to Sesame Street, can be seen in the Figure 6.17 below.

220 BSD Consultants Pty. Ltd. (2002), *2001 Perth Suburban Rail Patronage Count Report*, BSD Consultants Pty. Ltd., Perth

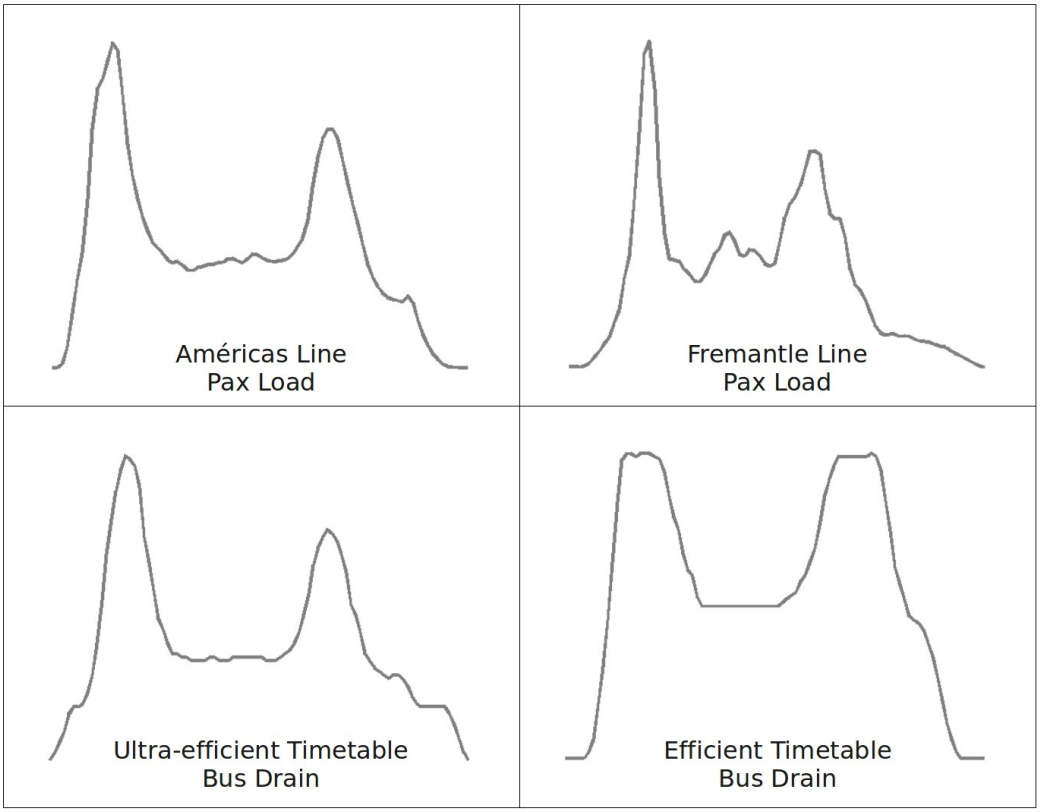


Figure 6.17: Three of these Things Belong Together

Passenger load curves are by their nature non-symmetrical and lacking in angular transitions and lengthy flat-zones, and so, as the key characteristic of ultra-efficient timetabling is the rapid tracking of passenger loads, the ultra-efficient curve's shape is of the same nature. Taking the ultra-efficient timetable development approach non-pseudo-live would not change this. The gross difference between the shapes of the efficient and ultra-efficient load curves would strongly suggest that substantial performance benefits would remain if an ultra-efficient timetable were constructed and assessed using a non-pseudo-live approach.

6.11 Operationalisation: Dragons Be Here

In chapter two of this thesis the similarities between the transit scheduling problem and the BRT timetabling problem were discussed. The description by a scheduler of one transit scheduling software tool as being a “fearsome beast” was noted, and the idea advanced that a primary task of this thesis would be to build a similarly fearsome beast to tackle the BRT timetabling problem; a small, prototype fearsome beast. In chapter four of this thesis it was discussed that the output from such a BRT timetabling software tool was likely to be ultra-complex, and that it would be customers not drivers who would be faced with this complexity. In this chapter, the performance of the Matilda software package, the small, prototype fearsome beast used to generate an ultra-efficient timetable for the Américas Line, has been presented and analysed, and, indeed, the output that has delivered that performance has turned out to be ultra-complex. Take, for example, just the express services for the two highest load time periods, 7:15 to 7:30 am and 7:30 to 7:45 am on the inbound line.

```
07:15 - 07:30 inbound

Services per
15-minutes        Stopping Pattern

5           -- -- -- -- 13 12 -- -- - - - - - - - - - 0
7           17 16 -- -- -- -- -- -- - - - - 5 - - - - 0
6           17 -- 15 -- 13 -- -- -- - - - - - 4 3 2 - 0
6           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


07:30 - 07:45 inbound

Services per
15-minutes        Stopping Pattern

6           -- -- -- -- 13 12 -- 10 - - - - - - - - - 0
6           17 -- -- 14 -- -- -- -- - - - - - - - - - 0
6           17 16 15 -- 13 -- 11 -- - - - - - 4 3 - - 0
5           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

Figure 6.18: Ultra-efficient Timetable Stopping Pattern Sample

There are six express services, in total, for the two 15-minute inbound periods shown above, each with a different stopping pattern. The efficient timetable for the Américas Line uses three different stopping patterns in total, the stopping patterns for express services 80, 100 and 120. In the above thirty-minute inbound time period alone, the ultra-efficient timetabling method is using twice the number of express stopping patterns. As previously mentioned, a full listing of the ultra-efficient timetable generated for the Américas Line is available in appendix two. This listing shows the full one-day ultra-efficient timetable making use of 136 total stopping patterns, 88 of which are unique.

In section 4.5, “Defining Ultra-efficient Timetabling”, a rough estimate was given of the size difference between a printed timetable for the existing May 2005 TransMilenio timetable and one for an equivalent ultra-efficient timetable. It was shown, on the basis of the comparative number of express services, that the ultra-efficient timetable would be approximately eighty times larger than the 2005 efficient timetable. With the ultra-efficient timetable development exercise now complete, this rough estimate, which was for the TransMilenio system as a whole, can be followed up by a similar example for just the Américas Line, one based on final data. The May 2005 efficient timetable uses three express stopping patterns, the stopping patterns for express services 80, 100 and 120. With the ultra-efficient timetable using 88 unique stopping patterns, a paper version of the ultra-efficient timetable would be approximately 29 times the size. The difference between the rough estimate of a timetable approximately eighty times larger, and the final magnitude of one approximately 29 times larger, is primarily due to the rough estimate

being based on what has turned out to be an overestimate of express service use. A visual representation of the difference in size between the May 2005 efficient and ultra-efficient timetables is shown in Figure 6.19 below.

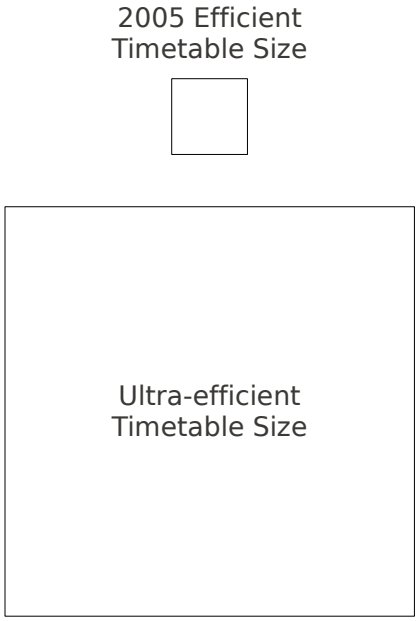


Figure 6.19: Efficient and Ultra-efficient Timetable Sizes

Consider the response of passengers, used to a timetable the size the one at the top of Figure 6.19, if they were to be presented with a timetable as greatly increased in size as the one at the bottom. Interestingly, in 2006, shortly after the May 2005 reference date for the data used in this thesis, TransMilenio S.A. did indeed increase the size of their timetables, with the number of weekday express services used on the Américas Line increasing from three to six, and the number on the system as a whole approximately tripling from 12 to 37.[221]

221 From a copy of the then current system timetable, which was obtained during the site visit in June 2006. The Américas Line express service numbers, those with unique stopping patterns, are designated on this timetable as B14, B52, C19, D50, F23 and F60.

The driving force behind these changes appears to have been a serious cost-control focus amongst senior management, meeting a 'can-do' attitude from the planners, in the context of a pressing need to expand the network.[222] There seems little to criticise here; having cost-focused management is entirely appropriate when running public infrastructure in a developing country, and who would not want an enthusiastic team of planners with a 'can-do' attitude. Focus and enthusiasm, combined with a pressing need to control costs so as to allow for expansion can, though, lead to problems if a balance is not maintained between the sophistication of timetables deployed and the sophistication of the passenger information system in place. A more sophisticated passenger information system was not deployed to complement the new more sophisticated TransMilenio timetables; paper timetables of greater complexity were simply produced.

In an highly innovative move, what was deployed in conjunction with the TransMilenio's new more complex paper timetables, was the TransMilenio S.A. planning department. They were sent out to the TransMilenio's major BRT stops to explain their new timetables to Bogotá's travelling public. It seems that Bogotá's travelling public did not immediately warm to the new more complex timetables, and expressed their feelings about the new timetables to the planners in full and frank terms. Translated into the management language criticised in section 2.2 of this thesis, "What is Bus Rapid Transit?", the task of explaining the new more complex timetables was described as being an extremely

222 Numerous observations and assessments in this paragraph and the following one are drawn from notes taken at a meeting between the author and Alejandro Niño of the TransMilenio S.A. Planning Department on the 7th of June 2006.

challenging one. Perhaps when Mr Wright chose to list “excellence in customer service” as a defining characteristic of BRT, again in section 2.2, he had such situations in mind.[223] For the case of explaining complex express patterns, over and over, to a travelling public familiar with an environment of robust public discourse about transportation matters,[224,225] a more apt choice might have been stoicism in service.

These deployment difficulties were caused by an approximate three-fold increase in the complexity of the TransMilenio BRT timetable, from 12 express services per weekday to 37. Figure 6.20 below, reproduces the Américas Line timetable-size depiction shown in Figure 6.19, with the more complex 2006 timetable added for comparison purposes. The approximate three-fold increase in the overall TransMilenio express levels has been used to construct the below 2006 timetable size, rather than the two-fold increase on the Américas Line.

223 Wright, L. (2003), *Bus Rapid Transit - A Global Review*, Institute for Transportation & Development Policy, New York, slide 3

224 Bogotá *Faces Second Day of Strike*, BBC Website, Available: http://news.bbc.co.uk/2/hi/americas/4970166.stm (Accessed 5 June 2006)

225 Bogotá *Public Transportation Collapses*, Prensa Latina Website, Available: http://www.plenglish.com/article.asp?ID=%7B0F6145EE-BEB3-4ABB-8219-7FF09AC15A44%7D&language=EN (Accessed 5 June 2006)

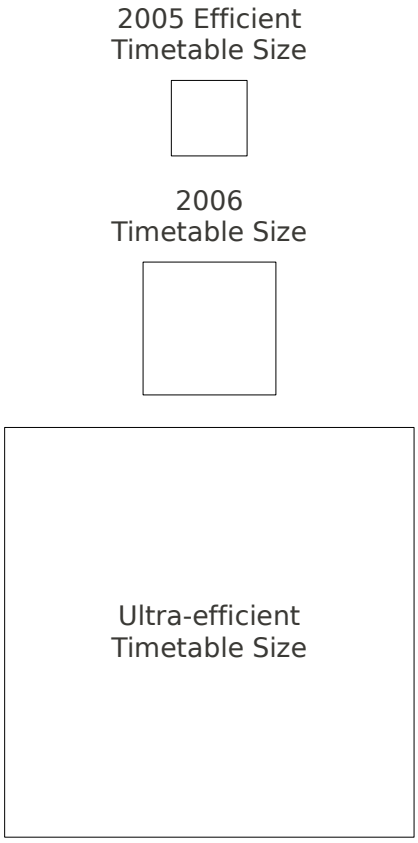


Figure 6.20: Efficient, 2006 and Ultra-efficient Timetable Sizes

The deployment history of the TransMilenio S.A.'s 2006 timetable, strongly suggests that this level of timetable complexity is pushing the upper-bounds with regards to how complex timetables can be, while still relying of paper timetables. As can be seen in Figure 6.20 above, the ultra-efficient timetable is approximately ten times more complex than the TransMilenio S.A.'s 2006 timetable.

This discontinuity between a complex version of an efficient BRT timetable and the ten times more complex ultra-efficient timetable, reinforces the labelling of this latter type of timetable as "ultra-efficient." The type of ultra-complex timetable developed for this thesis could, for example, have alternatively been labelled as either a "very-efficient" or a "highly-efficient" timetable. It is the case, though, that both the phrases

"very-efficient" and "highly-efficient" are almost entirely positive in nature, implying little or no negative connotations. For example, in common speech if a machine were said to be running very-efficiently, or a given process said to be highly-efficient, such circumstances would normally be seen as wholly positive.

Although the phrase "ultra-efficient" does have positive connotations, suggesting that things could not be running any more efficiently, it also has some negative connotations associated with it. The *Oxford English Dictionary* defines "ultra-", when used as a prefix, as "beyond; on the other side of" and "extreme; to an extreme degree."[226] In a similar vein, Roget's Thesaurus' list of similes for "ultra-" includes "drastic, excessive, immoderate" and "too much."[227] Both these dictionary definitions and thesaurus similes, at the very least, leave room for the possibility that an ultra-efficient process may be one that is efficient in excess of what is desirable. As discussed in this section, one way in which a timetable may be efficient in excess of what is desirable, is for it to be so complex that passengers cannot understand it. TransMilenio S.A. pushed the boundary of passenger timetable comprehension, under a regime of paper timetables, with a timetable ten times simpler than the ultra-efficient one developed here.

As was discussed in section 4.5, "Defining Ultra-efficient Timetabling", the ultra-efficient timetabling research presented in this thesis simply assumes the presence of a fully-automated passenger information system. Section 4.5 goes on to specify that a fully-automated

226 *AskOxford: ultra-*, AskOxford Website, Available: http://www.askoxford.com/concise_oed/ultrax?view=uk (Accessed 13 Febuary 2009)

227 *Ultra Synonym - Thesaurus.com*, Thesaurus.com Website, Available: http://thesaurus.reference.com/browse/ultra (Accessed 13 Febuary 2009)

passenger information system is one “where passengers have no real involvement in selecting their route, they simply arrive at their departure stop and a live passenger information display board tells them which bus they need to board.” Exploring the outer-edge of BRT timetabling efficiency, should not be taken as an unconditional recommendation to operators to take their systems up to that outer edge. To deploy an ultra-efficient timetable successfully, the use of a fully-automated passenger information system would be non-optional. TransMilenio S.A. found that deploying a timetable approximately one-tenth the complexity of an ultra-efficient timetable, required their staff to persevere through ‘highly challenging public education encounters.’ Deploying a timetable approximately ten times as complex as the one deployed by TransMilenio S.A. without a passenger information system of similar sophistication, would lead to more than some ‘challenging’ encounters; the efforts required would need to be positively heroic. Dragons Be Here.

Warning posted, as this section has shown, along with the dragons there is also a great deal of treasure to be had. The bluntest example of this is the approximate 40% reduction in bus kilometres under the ultra-efficient timetable, giving a saving of the order of US$47 million per year for the TransMilenio system as a whole. Even if, due to practical deployment issues, only half of that 40% efficiency gain could be realised, the saving would still be of the order of US$23 million per year. A sum of US$23 million buys a great deal of information technology with which to deploy a fully-automated passenger information system.

Finally, as an alternative to actually deploying an ultra-efficient timetable, at least in the short term, the option exists to use a version of

the Matilda software package developed for this thesis to calculate more efficient variants of ‘normal’ BRT timetables, ones that were no more complex than currently deployed timetables. One way of achieving this outcome would be to modify the Matilda software package so as to allow a maximum number of stopping patterns to be specified, and to reject any solutions beyond that level, no matter how efficient. In the test case for this thesis, the Américas Line, this would mean restricting the number of express services to three, to achieve a complexity match to the May 2005 efficient timetable. Limiting the number of express services the Matilda software package was ‘permitted’ to use, although an attractive option in many respects, would be a difficult change to implement technically. At the moment each time-slice is calculated completely independently, and the software is free to allocate any stopping pattern to each time-slice. To limit the number of express services used would require the software to mediate, in some fashion, between the competing needs of each time-slice, which would be a substantial and difficult to implement change.

A simpler way of using the software to develop more efficient variants of ‘normal’ BRT timetables, would be to convert the package into a tool suitable for use in current BRT timetable development procedures. The Matilda software package was designed to explore ultra-efficient timetabling, by examining passenger load in 15-minute time slices, and automatically matching efficient express service stopping patterns to those loads. The 15-minute time period used, though, is not ‘hard-wired’ into the Matilda software; any time duration from 15-minutes to a whole day could be assessed. By significantly increasing the length of the time

period used by the Matilda software package, highly-efficient stopping patterns for normal efficient BRT timetables, such as those used by the TransMilenio, could be automatically identified.

As mentioned in the introduction to this chapter, for each passenger load pattern, the Matilda software searches approximately 65 to 200 thousand express service stopping patterns looking for the most efficient option. It would seem inevitable that this automated approach would produce a more efficient outcome than any manual approach, such as the TransMilenio S.A.'s spreadsheet-assisted method of manually checking one stopping pattern at a time. It is an open question as to whether, with an automated tool at their disposal, TransMilenio S.A.'s planning department might have been able to deliver the efficiency gains yielded by their more complex 2006 timetable, without having to make it fully three times more complex.

Either deploying an ultra-efficient timetable itself or deploying an automated software tool to search through express patterns, would be significant undertakings. With regards to the primary option examined in this thesis, deploying an ultra-efficient timetable, the 'challenges' identified need to be viewed alongside the significant efficiency gains that have been discussed throughout this chapter.

This section has addressed the ninth and tenth research questions of this thesis. The ninth research question asked was whether, given the likely greater complexity of any ultra-efficient timetable, it would be possible to realistically deploy such a timetable without a fully-automated passenger information system? This question has been answered in the negative. The ultra-efficient timetable that has been produced here is

approximately ten times more complex than the 2006 TransMilenio paper-based timetable, a timetable whose complexity resulted in significant deployment difficulties.

The tenth, and final, research question asked was whether any of the techniques developed to produce ultra-efficient BRT timetables could be applied to improve the efficiency of current ‘normal’ paper-based BRT timetables, while keeping those timetables as paper-based timetables. This question has been answered in the positive. As this section has discussed, the technique of automatically searching through a very large number of express patterns could be applied to the significantly longer time durations used by ‘normal’ paper-based BRT timetables.

6.12 Conclusion

This chapter has presented the performance data from the development of an ultra-efficient timetable, and compared it to the present efficient timetabling method. The ultra-efficient timetabling method was shown to be tracking passenger load with far greater precision than the efficient method, and substantial efficiency gains were shown, specifically reduced bus fleet size, reduced bus kilometres, reduced bus time-in-service and higher bus occupancy. In terms of dollars, the efficiency gains obtained via ultra-efficient timetabling translated into a US$1.8 million reduction in capital costs, and a US$6.0 million reduction per year in weekday operating costs for the Américas Line.

The speed performance of the efficient and ultra-efficient timetabling methods were shown to be largely similar. While analysing the speed

performance of the ultra-efficient timetabling method, the Matilda software tool was utilised to critique the view that Rail Rapid Transit (RRT) technology is inherently faster than Bus Rapid Transit technology. In a like case, the London Underground, BRT was shown, with only modest infrastructure changes, to be able to match the speed performance of RRT.

The question of whether the performance results for the ultra-efficient timetable could be relied upon was then addressed, due to a concern over the possible impact of what has been referred to as a pseudo-live effect. An analysis of passenger curves from unrelated systems, showed very significant differences between the efficient and ultra-efficient timetables, in terms of their similarity to normal passenger load curve shapes. This analysis supported the conclusion that substantial performance benefits would remain if an ultra-efficient timetable were constructed using a non-pseudo-live approach.

Finally, issues relating to the operationalisation of an ultra-efficient timetable were discussed. The importance of matching passenger information system sophistication to timetable sophistication was emphasised, and the importance of deploying a fully-automated passenger information system along with an ultra-efficient timetable was reiterated. Some of the difficulties TransMilenio S.A. encountered, when they used paper timetables to deploy a timetable one-tenth as complex as the equivalent ultra-efficient timetable, were covered. The possibility of using the Matilda software, written to produce ultra-efficient timetables, to improve the efficiency of normal efficient timetables was discussed. Although operationalising the technology developed for this thesis, either

to improve existing BRT timetables or to deploy ultra-efficient BRT timetables, would present significant difficulties, it was highlighted that there were also significant benefits available to be realised.

< blank page >

< blank page >

< blank page >

< blank page >

< blank page >

Chapter 7

Conclusion

7.1 Thesis Summary

BRT systems differ from other transit systems, in that they commonly run express services side-by-side with an all-stop service. It is not unusual for high-load BRT systems to run multiple express services side-by-side with an all-stop service, such as the case of the Américas Line of the TransMilenio BRT which, in May 2005, ran up to three express services side-by-side with an all-stop service. Most transit systems can be timetabled by the application of the relatively simple maximum load procedure, discussed in chapter two. The running of express services side-by-side with an all-stop service, leads to the timetabling of BRT systems becoming a combinatorial problem, a far more complex class of problem than for normal timetabling.

The TransMilenio S.A. planners tackled this complex BRT timetabling situation using non-BRT-specific tools and techniques along with a spreadsheet-assisted method of manually checking one express service stopping pattern at a time. There is no evidence of a purpose designed BRT timetabling software tool being available, one built around a recognition of the problem's inherently combinatorial nature. As was discussed in chapter two, another area of the transit planning process, vehicle scheduling, is also a combinatorial problem. A number of purpose designed vehicle scheduling software tools, ones built around a recognition of *this* problem's inherently combinatorial nature, are available, and are in widespread use. By comparison to the manual

methods they have superseded, these tools have significantly increased the efficiency of vehicle scheduling. Drawing on the combinatorial similarity between vehicle scheduling and BRT timetabling, the proposition has been advanced that a combinatorial BRT timetabling software tool may be able to timetable BRT systems more efficiently than current methods.

The validity of this proposition has been tested by building a prototype combinatorial timetabling software tool, and testing it against a 'normal' BRT timetable, one developed using spreadsheet-assisted methods. The combinatorial timetabling tool that has been developed automatically searches through and assesses up to approximately 200 thousand express service stopping patterns for each 15 minute time slice, both inbound and outbound. This automated timetabling approach was tested against current methods by producing an alternative timetable for the TransMilenio BRT's Américas Line, using data from the period Monday the 23rd to Friday the 27th of May 2005. Identical physical and operational characteristics to the existing timetable were used for the alternative timetable, so as to allow for meaningful comparisons to be drawn between the two.

One explicit difference between the existing and alternative timetables, was in the area of their respective passenger information systems. The existing May 2005 timetable, referred to as an "efficient timetable", was developed to be deployed using paper-based timetables. By contrast, the alternative timetable, referred to as an "ultra-efficient timetable", has been developed on the assumption that it would be deployed using a fully-automated passenger information system, one able

to tell passengers at each station, minute-by-minute, which buses they need to board to reach their destination stations. The ultra-efficient timetable has been made as complex as necessary to service passenger load as efficiently as possible.

The process of producing an ultra-efficient timetable has required a significant number of research questions to be addressed, questions which were laid out in the introduction. As a starting point, for a thesis investigating the timetabling of BRT systems, it was necessary to ask:

> Is there a clear definition of what is, and what is not, a BRT system in the literature?

This question was answered in chapter two, in the negative. Section 2.2, "What is Bus Rapid Transit?", discussed the fact that authoritative sources gave conflicting, unclear or highly contestable definitions of BRT. The clearest definition of BRT found in the literature was from a Canadian Urban Transit Association paper, which defines BRT as "a rubber tired rapid transit system with running ways, stations, and all the other attributes normally associated with rail rapid transit."[228] A working definition of BRT was developed in section 2.2, which defined BRT as: a bus-based transit mode, capable of servicing high passenger loads, that at least partially operates on a bus-only roadway, that very often has at least partially enhanced station environments and that often features level boarding and pre-boarding ticketing.

228 McCormick Rankin Corporation (2004), *Bus Rapid Transit: A Canadian Industry Perspective*, Canadian Urban Transit Association, Toronto , p. 15

With a working definition of BRT settled upon, it was then possible to ask:

> Is there any theoretical evidence that current manual timetabling techniques are ill-suited to the task of producing highly efficient BRT timetables?

This question was also answered in chapter two, with the answer to the question depending on the type of BRT system. For BRT systems which do not run express services, the maximum load procedure method discussed in section 2.4, “An Overview of Transit Timetabling and Scheduling”, appears perfectly well-suited to the production of highly efficient BRT timetables. By contrast, for BRT systems which run express services, side-by-side with an all-stop service, there was clear theoretical evidence that current manual timetabling techniques are not well-suited to the task of producing highly efficient BRT timetables. In section 2.6, “High-load BRT Timetabling: A Special Case”, an ordinary length transit line, running only one express service side-by-side with an all-stop service, was shown to have over 30 million possible stopping patterns. Current methods of assessing this vast number of stopping patterns are essentially manual, and so only a tiny proportion of these stopping patterns are ever assessed for their potential efficiency.

Following up this theoretical assessment of the effectiveness of current BRT timetabling techniques, with a practical assessment of their effectiveness, involved analysing data from the TransMilenio BRT system. For this analysis to be conducted, the Américas Line of the TransMilenio

BRT system needed to be separated from the system as a whole, which made it necessary to ask:

> Can the data for one line of a BRT system be easily separated from the data for the entire system?

In chapter three is was shown that one line of a BRT system can be separated from the data for the entire system, but that this separation can only be achieved with difficulty. A significant proportion of chapter three was required to detail the process of separating the Américas Line of the TransMilenio BRT from the system as a whole.

Although a complicated task, separating of the Américas Line data made it possible to ask:

> Are current BRT systems timetabled in a highly efficient fashion?

This question was answered in chapter three, in the negative. In section 3.9, “Comparing the Bus Run and Passenger Load Data”, the bus drain levels for the current Américas Line timetable were shown to be only loosely tracking passenger load levels. Furthermore, for the majority of the day, the Américas Line was shown to be running significantly more buses, in proportion to passenger load, than during the time of the morning peak. This real-world efficiency assessment, confirms the previous theoretical assessment that BRT systems are not currently timetabled in a highly efficient fashion.

With evidence having been presented that ‘efficiency room’ exists above current timetabling practices, room for an ultra-efficient class of timetabling to occupy, the question of just how efficient ultra-efficient timetables are, was addressed by asking:

> Is it possible to generate optimally efficient timetables for BRT systems.

This question was answered in section 4.3, “A Practical Characteristic – Computational Optimisation”, with the answer to the question depending on the type of BRT system. For BRT systems which do not run express services, it appears that the application of the maximum load procedure method will yield an optimal timetable. By contrast, for BRT systems which run express services, side-by-side with an all-stop service, it is not always possible to generate an optimally efficient timetable in a realistic period of time. In section 4.3 it was shown that the three express service case of the Américas Line could be timetabled in approximately 281 trillion different ways. A rough estimate was made of how long it would take to generate a full-days optimal timetable for the Américas Line on a normal computer, which yielded a figure of approximately 320 years. With the demonstration that BRT systems cannot always be optimally timetabled, it was confirmed that ultra-efficient timetabling is not just another term for optimally-efficient timetabling.

Identifying just where ultra-efficient timetabling is positioned, in the ‘efficiency room’ that exists between current timetables and optimally-

efficient timetables, required the concept of ultra-efficient timetabling to be clearly defined. It was therefore necessary to ask:

> Is it possible to develop a definition of ultra-efficient BRT timetabling, one that clearly differentiates it from current BRT timetabling?

This question was answered in chapter four, in the positive. In section 4.5, “Defining Ultra-efficient Timetabling”, a clear definition of ultra-efficient timetabling was developed, one based on the way in which passengers interact with a BRT system’s timetable. A BRT system was said to have been timetabled in an ultra-efficient manner if: its services change so rapidly that passengers can no longer be reasonably expected to determine by themselves which service they need to board. With a clear definition of what an ultra-efficient timetable is, it was then possible to develop a method with which to actually produce such a timetable.

At the centre of the method developed to produce ultra-efficient timetables, was the essential step of judging the performance of different express service stopping patterns. To conduct this judging process, required the development of a metric with which to be able to say that one express service stopping pattern choice was better than another. To be able to develop such a metric, it was necessary to ask:

> Can a simple and effective metric to judge the efficiency of BRT express service stopping patterns be easily devised?

In chapter five is was shown that a simple and effective metric to judge the efficiency of BRT express service stopping patterns could be developed, but that the development of such a metric involved significant difficulties. Section 5.7, “Making the Runs for One Direction and Time Period Only”, discussed the fact that the first, carefully considered, metric developed to judge the efficiency of BRT express service stopping patterns exhibited highly undesirable short-trip avoidance behaviour. It was therefore necessary to develop a second metric, one based on minimising the total number of intermediate stops that express services stopped at. This metric, embeded at the centre of a purpose designed software tool, allowed for the production of an ultra-efficient timetable.

By comparing the ultra-efficient timetable that has been produced here to the existing efficient timetable, it was then possible to address the question:

> Can a BRT timetable which varies its service frequencies and stopping patterns at the same pace that passenger load levels and patterns change, significantly outperform a normal BRT timetable?

This question was answered in chapter six, in the positive. In section 6.3, “Bus Drain”, bus fleet size under the ultra-efficient timetable was shown to be 9% lower than for the existing timetable, indicating an estimated capital cost saving for the Américas Line of US$1.8 million. In section 6.4, “Bus Distance” bus kilometres travelled under the ultra-efficient timetable were shown to be 40% lower than for the existing timetable,

indicating an estimated annual operating cost saving for the Américas Line of US$6 million. These efficiency gains can clearly be tied to the fact that the ultra-efficient timetable is tracking passenger load far more precisely than the efficient timetable. Indeed, as shown in section 6.9, “Comparing the Bus Run and Passenger Load Data”, for much of the day the bus drain curve for the ultra-efficient timetable appears as though it has been draped over the passenger load curve.

In the introduction, the above question was identified as the central question to be answered by the research presented in this thesis. This research question was also rephrased as a thesis, with the thesis being whether: a BRT timetable which varies its service frequencies and stopping patterns at the same pace that passenger load levels and patterns change, can significantly outperform a normal BRT timetable. This thesis has been confirmed.

Not only has the production of an ultra-efficient timetable allowed its performance to be assessed, it has also made it possible to ask:

> Given the likely greater complexity of any ultra-efficient timetable, would it be possible to realistically deploy such a timetable without a fully-automated passenger information system?

This question was also answered in chapter six, in the negative. In section 6.11, “Operationalisation: Dragons Be Here”, the ultra-efficient timetable produced for the Américas Line was shown to have 88 unique stopping patterns, compared to the efficient timetable’s three, making it

approximately 29 times as complex. In 2006 the TransMilenio S.A. faced significant difficulties deploying a timetable approximately three times as complex as the 2005 efficient timetable, due to issues some passengers were having understanding a paper timetable of that complexity. This deployment experience strongly suggests that the level of complexity of the 2006 TransMilenio timetable is either at or near the upper-bounds of how complex timetables can be, while still relying on paper timetables. The ultra-efficient timetable that has been produced here is approximately ten times more complex than the 2006 TransMilenio timetable.

Finally, the production of an ultra-efficient timetable has made it possible to ask:

> Could any of the techniques developed to produce ultra-efficient BRT timetables be applied to improve the efficiency of current 'normal' paper-based BRT timetables, while keeping those timetables as paper-based timetables?

This final research question is the third question to be answered in chapter six, with the answer being in the positive. At the end of section 6.11, "Operationalisation: Dragons Be Here", the option of using a variant of the ultra-efficient timetabling software package developed here, to improve the efficiency of existing efficient BRT timetables, was discussed. Perhaps the most interesting outcome from the research presented in this thesis is that, as the best current stand-in for a theoretical optimum, it indicates that BRT systems appear to be being

timetabled at a substantial distance away from their theoretical efficiency limit. The purpose of the research presented here, has been to move closer to that theoretical efficiency limit, without regard to whether the resulting timetable could be deployed using paper timetables. In pursuit of this goal, a software tool has been built that searches through approximately 65 to 200 thousand express service stopping patterns, looking for the most efficient option. For the purposes of producing an ultra-efficient timetable, individually tailored express patterns have been made for each 15-minute time slice, both inbound and outbound. There is no reason, though, that the general approach of automatically searching through a very large number of express patterns could not be applied to the significantly longer time durations used by 'normal' paper-based BRT timetables. It would seem inevitable that an automated approach would produce a more efficient outcome than any manual approach, such as the TransMilenio S.A.'s spreadsheet-assisted method of manually checking one stopping pattern at a time.

The research presented in this thesis has investigated both the potential benefits and the potential risks of ultra-efficient Bus Rapid Transit timetabling. The primary benefit of ultra-efficient Bus Rapid Transit timetabling is the ability to produce more efficient timetables, whereas the primary risk is deploying such timetables using passenger information systems of insufficient sophistication to support them. Further to this primary risk, any attempt to operationalise the research presented here, would no doubt face all the difficulties and problems normally encountered when trying to convert preliminary research into something that can realistically be deployed.

In particular, deploying an ultra-efficient timetable would involve information technology becoming an essential *operational* component of a BRT system. Although information technology is already an essential component in the planning of current BRT timetables, the comparative simplicity of such timetables allows information technology to be an optional component with regards to their deployment. Given the comparative simplicity of current BRT timetables, current BRT services can continue operation even if any information technology being used to improve operational performance fails. This would not be the case for an ultra-efficient timetable; for an ultra-efficient timetable if the operational information technology failed so would the operation of the BRT system. Consequently, a different attitude would be required by operators, one that viewed information technology as being as essential to a BRT system's operation as drivers or diesel. The difficulties and problems involved in this shift would largely be related to the mind-set of operators; as mentioned in section 6.11, "Operationalisation: Dragons Be Here", the potential cost savings from an ultra-efficient timetable would almost certainly pay for the required information system.

In the corporate world, whose interesting use of language was examined at the start of this thesis, there are no such difficulties or problems, there are only *challenges*. The academic equivalent of facing the world with such beaming optimism might be to say there are no risky research projects, only *exciting* ones. Deploying an ultra-efficient Bus Rapid Transit timetable would be a most exciting project.

7.2 Suggestions for Further Research

Beyond the already discussed options for operationalising the research presented here, there are a number of other promising directions for further research. One area in which further research could be pursued, would be to improve the approach used to handle the problem of intractability; the problem of timetable generation not completing within a reasonable period of time. The research presented in this thesis dealt with this problem by selecting a relatively short line as a test case, the 18 stop Américas Line, and by sequencing the generation of express services, rather than generating those express services simultaneously. As discussed in section 1.3, “Scope and Approach of the Research”, the sequencing of the generation of express services reduced the number of stopping patterns that needed to be assessed, for the most computationally intensive three express case, from approximately 281 trillion possible timetabling choices, to approximately 200 thousand options. Although searching approximately 200 thousand options is better than manually assessing one possibility after another, reviewing a substantially larger proportion of the full set of approximately 281 trillion possible timetabling options, in some non-intractable fashion, would be better still.

As discussed in section 2.5, “Transit Scheduling and the Combinatorial ‘Explosion’”, various highly sophisticated techniques have been developed that, in a reasonable period of time, yield very good, rather than optimal, solutions to intractable problems. As further discussed in section 4.3, “A Practical Characteristic – Computational Optimisation”, these techniques, developed by the operations research

discipline, were not applied in this thesis due to a need to contain the scope of the research, not due to any negative assessment of their likely utility. The stopping pattern search techniques applied in this thesis were the simplest effective non-intractable techniques that could be devised, not the most efficient ones. It seems all but inevitable that the application of the greatly more sophisticated techniques developed by the operations research discipline, would enable the production of a more ultra-efficient BRT timetable than the one presented here.

In a yet to be published paper, *Design of Express Services for an Urban Bus Corridor with Capacity Constraints*,[229] operations research techniques are use to assess the efficiency of express service stopping patterns for a bus corridor. This paper selects "a set of itineraries that appear to be *a priori* attractive, each one specifying a sequence of stops at which passengers can board and alight" and develops an operations research model that, for each itinerary, outputs "the frequencies of service and bus sizes to be used."[230] This research differs in some significant ways from the research presented here. The key difference is that the express service stopping patterns are selected *a priori*, rather than being searched for. Furthermore, the bus size is an output of the calculations, rather than fixed as it has been in the research presented here. Finally, there is no evidence that multiple simultaneous express services are handled, whereas in the research here they have been

229 Leiva, C., Muñoz, J. C. and Giesen, R. (2009), *Design of Express Services for an Urban Bus Corridor with Capacity Constraints*, Note: This paper by The Department of Transport Engineering and Logistics of the Pontificia Universidad Católica de Chile is, as of September 2009, in the second round referee stage for publication in Transportation Research B.

230 op. sit., Quoted text from Introduction section. (Note: Page numbers not given on this pre-publication document.)

explicitly catered for. These differences noted, this paper nevertheless demonstrates that operations research techniques can be effectively applied to assess at least the single express service case. Combining such operation research techniques with the automated express pattern searching techniques developed here, may well yield fruitful results. To summarise, applying the techniques of the operations research discipline to the problem of ultra-efficient BRT timetabling, represents a promising line of future research.

Another area in which research could be pursued to improve the efficiency of ultra-efficient timetables, is with regards to the sections of a BRT line that express services run along. The 2005 efficient timetable ran two express services along the whole length of the Américas Line, and turned one express service around four stops short of the far end of the Américas Line, at the Banderas stop. The ultra-efficient timetable developed here mirrored the turn-around points of the efficient timetable, with no attempt being made to investigate whether other turn-around points, at either end of the line, might have yielded a more efficient outcome. In a similar manner to the way in which all, or a large number of stopping patterns can be cycled through, looking for an efficient outcome, the many options as to which section of a line a given express service is to be run along, could also be cycled through and assessed. Investigating this approach to turn-around point selection, represents another promising line of future research.

Beyond improving the efficiency of ultra-efficient timetables, further research could be conducted to expand the applicability of the research presented here to a wider selection of transit modes. The ultra-efficient

timetable developed here, assumes that there are passing lanes at stops, as indeed there are on the TransMilenio BRT system, and therefore that bus services can pass one another. Consequently, as constructed, the type of ultra-efficient timetables developed here could not be run on any transit mode without passing lanes at every stop, which is most or all rail systems, and many bus-based systems. Some of these rail and bus-based systems would have the load to support at least one express service, if not more, and so being able to produce non-passing-lane ultra-efficient timetables would have utility. One way in which non-passing-lane ultra-efficient timetables might be produced, would be to filter out all stopping pattern options that involve one vehicle passing another. Even if this required 99.9999% of options to be discarded, for the three express case on the Américas Line, which has approximately 281 trillion possible stopping patterns, that would still leave approximately 281 million usable options with which to assess efficiency. It might be the case that ultra-efficient timetables could be developed for non-passing-lane transit modes, such as commuter railways.

Finally, beyond issues of increasing the efficiency or widening the applicability of ultra-efficient timetables, the pre-determined nature of the ultra-efficient timetables could be reviewed. As has been discussed, like the efficient timetable, the ultra-efficient timetable produced here is a pre-determined, rather than live, timetable; the ultra-efficient timetable is merely a very complicated pre-determined timetable. So complicated indeed, that deploying an ultra-efficient timetable requires abandoning the notion that passengers should be part of the decision making process about their route. As has been emphasised in this thesis, an ultra-

efficient timetable would need to be deployed in conjunction with a fully-automated passenger information system. Once this change to a fully-automated passenger information system has occurred, and passengers have ceased to be a part of the decision making process about their route, the obvious next step is to do timetabling live.

One view of both the efficient and ultra-efficient BRT timetables presented in this thesis is that they are both rather primitive, in that they both treat the timetabling problem as a puzzle to be solved, ahead of time, with the ‘puzzle solution’ then being deployed in the form of a timetable, paper or electronic. The alternative to this approach is to timetable a BRT system live, by assessing the current passenger loads on the line, and organising the current bus resources on the line, minute-by-minute, to best service those loads. Any such live timetabling of BRT systems would require an automated way of selecting express service stopping patterns, something achieved, in prototype form, in this thesis. In terms of sophistication, a live timetabling system would go far beyond the pre-determined ultra-efficient timetable research presented here, into the realm of producing optimally-efficient timetables. The production of optimally-efficient timetables would require timetabling to be treated not as a puzzle to be solved, but as a state to be managed.

< blank page >

< blank page >

< blank page >

< blank page >

< blank page >

Bibliography

Offline

Advani, M., Tiwara, G. (2006), *Review of Capacity Improvement Strategies for Bus Transit Service*, Indian Journal of Transport Management, Volume 30, Number 4

Avis, D., Hertz, A., Marcotte O. (Editors) (2005), *Graph Theory and Combinatorial Optimization* , Springer Science+Business Media, Inc., New York

Blythe, P., et al. (2000), *ITS Applications in Public Transport: Improving the Service to the Transport System*, Journal of Advanced Transportation, Vol. 34, No. 3, Fall 2000

Bunte, S., Kliewer, N., Suhl L. (2006), *An Overview on Vehicle Scheduling Models in Public Transport*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds

Bussiecky, M. R., Zimmermann, U. T. (1996), *Combinatorial Optimization Methods for Optimal Lines in RealWorld Railway Systems* (Working Paper), Abteilung fur Mathematische Optimierung, Technische Universität Braunschweig, Germany

Cain, A. (Principal Investigator) (2006), *Applicability of Bogotá's TransMilenio BRT System to the United States - Final Report May 2006*, Federal Transport Administration, Department of Transportation, United States of America, Washington, D.C., U.S.A.

Canepa, B. (2007), *Bursting the Bubble: Determining the Transit-Oriented Development's Walkable Limits*, Transportation Research Record, Volume 1992

Ceder, A. (2003), *Public Transport Timetabling and Vehicle Scheduling*, In: Lam, W.H.K., Bell, G.H., *Advanced Modeling for Transit Operations and Service Planning*, Elsevier Science Ltd, Pergamon

Ceder, A., Stern, H. I. (1981), *Deficit Function Bus Scheduling with Deadheading Trip Insertions for Fleet Size Reduction*, Transportation Science, Volume 15, Number 4, November 1981

Daduna, J. R., Schneidereit, G., Voß, S. (2006), *Passenger Information Systems in Public Mass Transit: Where We Are and Where We Go*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds

Demery, L. W. Jr. (2002), *Bus Rapid Transit in Curitiba, Brazil - An Information Summary*, Publictransit.us (formally Carquinez Associates), Vallejo, California

Desaulniers, G. (2002), *Bus and Driver Scheduling in Urban Mass Transit Systems*, Slides from a presentation given at: Travel and Transportation Workshop, Institute for Mathematics and its Applications, Minneapolis, November 13, 2002

Dunning, R. A., and Richert, T. M. (2001), *Applying Lessons from Lean Production Theory to Transit Planning*, Paper presented at: ASCE 8th International Conference on Automated People Movers July 2001

Eberlein, X. J., Wilson, N.H.M. and Barnhart, C. (1998), *The Real-time Deadheading Problem in Transit Operations Control*, Transportation Research. Part B: Methodological, Volume 32, Number 2

Eranki, A. (2004), *A Model to Create Bus Timetables to Attain Maximum Synchronization Considering Waiting Times at Transfer Stops*, Masters Thesis, University of South Florida, Florida

Furth, P. G. (1989), *Updating Ride Checks with Multiple Point Checks*, Transportation Research Record, Volume 1209

Grötschel, M., Lovász. L. (1995), *Combinatorial Optimization*, In: Graham, R., Grötschel, M., Lovász, L. (Editors), *Handbook of Combinatorics*, Volume 2, Chapter 28, pp. 1541-1597, The MIT Press, North Holland

Hawken, P., Lovins, A. and Lovins, L. H. (1999), *Natural Capitalism: Creating the Next Industrial Revolution*, Little, Brown and Co., Boston

Hensher, D. (1999), *A Bus-Based Transitway or Light Rail? Continuing the Saga on choice versus Blind Commitment*, Road and Transport Research, Vol. 8, No. 3, September 1999

Hillier, F. S., Lieberman, G. J. (2007), *Introduction to Operations Research (8th Edition)*, McGraw Hill Higher Education, London

Holdsworth, N., Enoch, M. P., Ison, S. G. (2007), *Examining the Political and Practical Reality of Bus-based Real Time Passenger Information*, Transportation Planning and Technology, Volume 30, Issue 2 & 3 April 2007, pp. 183-204

Homer, S., Selman, A. L. (2001), *Computability and Complexity Theory*, Springer-Verlag, New York

Huisman, D. (2004), *Integrated and Dynamic Vehicle and Crew Scheduling*, Ph.D. Thesis, Erasmus University Rotterdam, The Netherlands

International Energy Agency (2002), *Bus Systems for the Future: Achieving Sustainable Transport Worldwide*, Paris, France

Kenworthy, J. (2007), *Making Connections: Growing Public Transport's Market Share in Switzerland Through a More Customer and Information-Oriented Approach* (A Report to the Swiss Federal Railways), Murdoch University, Perth

Leiva, C., Muñoz, J. C. and Giesen, R. (2009), *Design of Express Services for an Urban Bus Corridor with Capacity Constraints*, Note: This paper by The Department of Transport Engineering and Logistics of the Pontificia Universidad Católica de Chile is, as of September 2009, in the second round referee stage for publication in Transportation Research B.

Levinson, H., Zimmerman, S., et al. (2003), *Transit Cooperative Research Program Report No. 90: Bus Rapid Transit. Volume 1: Case Studies in Bus Rapid Transit*, Transportation Research Board, Washington, D.C., U.S.A.

Levinson H. S., Zimmerman S., et al. (2004), *Transit Cooperative Research Program Report No. 90: Bus Rapid Transit. Volume 2: Implementation Guidelines*, Transportation Research Board, Washington, D.C., U.S.A

Li, H., Bertini, R. L. (2009), *Assessment of Optimal Bus Stop Spacing Model Using High Resolution Archived Stop-Level Data*, In: Transportation Research Board, *Transportation Research Board 88th Annual Meeting Compendium of Papers DVD*, Washington, D.C., U.S.A.

Liebchen, C., Stiller, S. (2006), *Delay Resistant Timetabling*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds

Maxwell, R. R. (1999), *Intercity Rail Fixed-Interval, Timed-Transfer, Multihub System: Applicability of the Integraler Taktfahrplan Strategy to North America*, Transportation Research Record, Volume 1691

McCormick Rankin Corporation (2004), *Bus Rapid Transit: A Canadian Industry Perspective*, Canadian Urban Transit Association, Toronto

Mees, P. (2000), *A Very Public Solution: Transport in the Dispersed City*, Melbourne University Press, Victoria, Australia

Menckhoff, G. (2005), *Latin American Experience with Bus Rapid Transit*, Paper presented at: Annual Meeting – Institute of Transportation Engineers, Melbourne, August 10, 2005

Miller, M. A., Yin, Y., Balvanyos, T., Ceder, A. (2004), *Framework for Bus Rapid Transit Development and Deployment Planning* (Research Reports: Paper UCB-ITS-PRR-2004-47), California Partners for Advanced Transit and Highways, Berkeley, California, U.S.A.

Mishalani, R. G., McCord, M. M., et al. (2006), *Passenger Wait Time Perceptions at Bus Stops*, Volume 9, Number 2, 2006

Neff, J. (2008), *2008 Public Transportation Fact Book (59th Edition)*, American Public Transportation Association, Washington, D.C., U.S.A

Nelson, J. D. (1995), *The Potential for Real-Time Passenger Information as Part of an Integrated Bus-Control/Information System*, Journal of Advanced Transportation, Vol. 29, No.1, Spring 1995

Panneerselvam, R. (2002), *Operations Research*, Prentice Hall of India, New Delhi

Papadimitriou, C. H. (1982), *Combinatorial Optimization : Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, N.J., U.S.A.

Planning and Transport Research Centre (2004), *Perth's South West Metropolitan Railway: Balancing Benefits and Costs*, Planning and Transport Research Centre, Perth, Australia

Rattadilok, P., Kwan, R. S. K. (2006), *Dynamically Configured λ-opt Heuristics for Bus Scheduling*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds

Samuel, P. (2002), *Busway vs. Rail Capacity: Separating Myth from Fact*, Policy Update 16, Reason Public Policy Institute, Los Angeles

Shih, M-C, Mahmassani, H. S. (1994), *Vehicle Sizing Model for Bus Transit Networks*, Transportation Research Record, Volume 1452

Souai, N., Teghem, J. (2006), *Hybridizing the Genetic Algorithm and the Simulated Annealing for the Airline Crew Rostering Problem*, Paper presented at: The 10th International Conference on Computer-Aided Scheduling of Public Transport, CASPT 2006, Leeds

Transportation Research Board (2003), *Transit Capacity and Quality of Service Manual (2nd Edition)*, Transportation Research Board, Washington, D.C., U.S.A.

UK Real Time Information Group (2005), *RTIG Monitoring: Enablers and Blockers to Rollout of RTI Systems*, UK Real Time Information Group, Guildford, United Kingdom

UK Real Time Information Group (2005), *The Integration of RTI into Bus Fleet Management*, UK Real Time Information Group, Guildford, United Kingdom

van Nes, R., Bovy P. H. L. (2000), *Importance of Objectives in Urban Transit-Network Design*, Transportation Research Record, Volume 1735

Vuchic, V. R. (2007), *Urban Transit Systems and Technology*, John Wiley & Sons, New Jersey

Watson, D. (2003), *Death Sentences: How Cliches, Weasel Words and Management-Speak Are Strangling Public Language*, Random House Australia, Sydney

Wiener, R., Lidor, G. (1978), *Passenger Utilization of Local vs Express Trains for a New York City Subway Line: A Case Study*, Transportation Research Record, Volume 662

Wren, A., Fores, S., et al. (2003), *A Flexible System for Scheduling Drivers*, Journal of Scheduling, Volume 6

Wright, L., Hook, W. (Editors) (2007), *Bus Rapid Transit Planning Guide*, Institute for Transportation & Development Policy, New York

Online

ACS Interdisciplinary Opportunities Initiative Definition, Associated Colleges of the South Interdisciplinary Opportunities Initiative Website, Available: http://www.colleges.org/newmodels/interdisciplinary/definition.html (Accessed 3 February 2005)

Bogotá, What's Next?: by Dario Hidalgo (2006)*, EMBARQ: The World Resources Institute Centre for Sustainable Transport Website, Available: http://www.embarq.wri.org/documentupload/WRI%20WB%20Jan%202006%20Dario%20Hidalgo.pdf (Accessed 17 February 2009)*

BRT Poised for Take-Off in China: by The Institute for Transportation & Development Policy (2005), The Institute for Transportation & Development Policy Website, Available: http://www.itdp.org/STe/ste19/brt.html (Accessed 24 January 2006)

BRT vs Rail Capacity (2003), Coalition for Effective Transportation Alternatives Website, Available: http://www.effectivetransportation.org/docs/BRTvsRailCapacityHarkness.pdf (Accessed 14 July 2003)

Bus Tops - Bus Rapid Transit Systems in South America are Changing the Faces of their Cities: by Shubhabrata Marmar (2006), Business Standard Motoring Website, Available: http://www.bsmotoring.com/2006/apr08_1.htm (Accessed 5 September 2006)

Designing Calm Technology: by Mark Weiser and John Seely Brown. (1995), Palo Alto Research Center Website, Available: http://nano.xerox.com/weiser/calmtech/calmtech.htm (Accessed 17 Febuary 2009)

Flexibility in Highway Design – Chapter 3 – FHWA (2003), U.S Department of Transport Federal Highway Administration Website, Available: http://www.fhwa.dot.gov/environment/flex/ch03.htm (Accessed 12 September 2003)

HASTUS: An Integrated Solution for Transit Scheduling and Operations (2007), Giro Company Website, Available: http://www.giro.ca/docs/public/pdf/hastus/en/GIRO-MKT-HASTUS-PST-ARCHE-20070322.pdf (Accessed 7 February 2009)

HASTUS: Transit Scheduling and Operations (2008), Giro Company Website, Available: http://www.giro.ca/docs/public/pdf/hastus/en/GIRO-MKT-HASTUS-PST-HASTUSE-20081124.pdf (Accessed 13 February 2009)

How to Optimise Code (2008), Programming for Scientists Website, Available: http://www.programming4scientists.com/2008/10/how-to-optimise-code/ (Accessed 19 February 2009)

Key Facts, Transport for London Website, Available: http://www.tfl.gov.uk/corporate/modesoftransport/londonunderground/1608.aspx# (Accessed 19 February 2009)

Linear Programming: A Concise Introduction: by Thomas S. Ferguson (2008), UCLA Department of Mathematics Website, Available: http://www.math.ucla.edu/~tom/LP.pdf (Accessed 22 January 2008)

Optimize Loops, Aivosto Website, Available: http://www.aivosto.com/vbtips/loopopt.html (Accessed 19 February 2009)

Progress in Solving Large Scale Multi-depot Multi-vehicle-type Bus Scheduling Problems with Integer Programming: by Uwe H. Suhl, Swantje Friedrich and Veronika Waue (2007), Association for Information System Electronic Library Website, Available: http://aisel.aisnet.org/cgi/viewcontent.cgi article=1080&context=wi2007 (Accessed 3 March 2009)

Q&A With Darío Hidalgo: by Erico Guizzo (2009), IEEE Spectrum Website, Available: http://www.itdp.org/STe/ste19/brt.html (Accessed 17 January 2009)

The Message from Bogotá (2003), Bristol Cycling Campaign Website, Available: http://www.bristolcyclingcampaign.org.uk/tbc/2003/summer/bogota.htm (Accessed 17 January 2009)

Towards a Better BRT Taxonomy: by Dario Hidalgo, The City Fix Website, Available: http://thecityfix.com/towards-a-better-brt-taxonomy/ (Accessed 17 January 2009)

TransMilenio: by Enrique Lillo, Steer Davies Gleave (2005), INRO Website, Available: http://www.inro.ca/en/pres_pap/european/eeug02/lillo.ppt (Accessed 29 May 2005)

TransMilenio Busway-Based Mass Transit, Bogotá, Colombia, The World Bank Website, Available: http://siteresources.worldbank.org/INTURBANTRANSPORT/Resources/Factsheet-TransMilenio.pdf (Accessed 17 January 2009)

TransMilenio's Contributions to the Development of Bus Rapid Transit Systems: by Darío Hidalgo Guerrero, Ph.D., Bogotá Lab Website, Available: http://www.bogotalab.com/articles/transmilenio.htm (Accessed 26 May 2008)

Why Is TransMilenio Still So Special? (2008), The City Fix Website, Available: http://thecityfix.com/why-is-transmilenio-still-so-special/ (Accessed 17 January 2009)

< blank page >

< blank page >

< blank page >

< blank page >

< blank page >

Appendix One

Matilda Source Code

Notes

1. The source code presented in this appendix is based on source code that has been previously submitted for assessment.[231] Approximately 20% of the source code in this appendix has been previously submitted for assessment. (Inclusion of such material checked with Thesis Committee Chairperson, Associate Professor Steve Basson, before thesis submission.)

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * main.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * main.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * main.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include <iostream>

#include <cmath>

#include <assert.h>

#include <algorithm>
using namespace std;

#include "line.h"
#include "run.h"
#include "pax_group.h"
#include "load.h"
#include "link.h"
#include "convert.h"
#include "combination.h"
#include "utility.h"


typedef pair<long,double> pair_long_double_T;
```

231 Bradley, M. (2003), *Flexways: The Advantages of a Tactical Imbalance in Transport Planning* (Honours Thesis), Murdoch University, Perth

```
//
// MakeRunGroups
//

void
make_one_run_group( int                 iTimeSlice,
                    EDirection          iDirection,
                    float               iNumberOfRuns,
                    vector<int>         iStops,
                    vector<int>         iLinks,
                    float               ioLinkLoads[k_links_size],
                    vector<int>         iLinksSpans[k_links_size],
                    vector<cRun>&       oRuns,
                    ostringstream&      oTextFeedback )
{
    // isolate and scale loads
    float the_load_for_one_run[k_links_size];
    ZeroLinkArray(the_load_for_one_run);
    for_all( vector<int>, iLinks, link )
        the_load_for_one_run[*link] = ioLinkLoads[*link]/iNumberOfRuns;

    // for all runs
    for( int run = 0; run < iNumberOfRuns; run++ )
    {
        // calculate centre time
        cInstant centre_time = cInstant( TimeSliceDuration(iTimeSlice).End() -
                                ( float(run) * (k_time_slice_length/iNumberOfRuns) ) );

        // make a run
        cRun the_run = cRun( iDirection, centre_time, (k_time_slice_length/iNumberOfRuns),
                             iStops, the_load_for_one_run );

        oRuns.push_back( the_run );

        // output feedback
        oTextFeedback    << the_run.Time(eFrom).RoundToSeconds() << space
                         << the_run.Time(eTo).RoundToSeconds() << space
                         << iNumberOfRuns << space
                         << the_run.Duration().Width().RoundToSeconds() << space
                         << round(the_run.Occupancy()*100.0) << space
                         << round(PeakLoad(ioLinkLoads, iLinksSpans,iLinks)/iNumberOfRuns)
                         << space
                         << PrettyPrintStops(iStops)
                         << endl;
    }
    oTextFeedback << endl;

    // zero the serviced loads
    for_all( vector<int>, iLinks, link )
        ioLinkLoads[*link] = 0;
}

void MakeRunGroups()
{
    ostringstream oss_log;

    cout << "start pre-calculations" << endl;

    // list each link's spans
    vector<int> links_spans[k_links_size];
    for_all_stops_from
    {
        for_all_stops_to
        {
            int min_stop = min(from,to);
            int max_stop = max(from,to);

            for( int span = min_stop; span < max_stop; span++ )
                links_spans[Link(from,to)].push_back(span);
        }
    }

    vector<int> links_far_all_stops_pos[EDirectionSize];
    for_all_links
        if( cLink(link).From() != cLink(link).To() )
            links_far_all_stops_pos[cLink(link).Direction()].push_back(link);
```

```
vector<int> stops_far_all_stops;
for_all_stops
    stops_far_all_stops.push_back(stop);

cout << "end pre-calculations" << endl;

//WriteFile( "oss_log.txt", oss_log.str() );

vector<cRun> runs;

// get loads and offset to line centre
cout << "get loads" << endl;
float from_to_loads[k_time_slices_size][k_stops_size][k_stops_size];
ReadLoads( from_to_loads, "loads.txt" );
OffsetLoadsToLineCentre( from_to_loads );

// convert from-stop/to-stop loads to link loads
float link_loads[k_time_slices_size][k_links_size];
for_all_time_slices
    for_all_stops_from
        for_all_stops_to
            link_loads[time_slice][Link(from,to)] = from_to_loads[time_slice][from][to];

// print total load
float total_load = 0;
for_all_time_slices
    total_load += TotalLoad( link_loads[time_slice] );
cout << "total load " << space << total_load << endl;

// ***
// *** pre-calculate runs and channels
// ***

float    num_runs[EDirectionSize][k_time_slices_size][ENearFarSize];
float    num_channels[EDirectionSize][k_time_slices_size][ENearFarSize];

for_both_directions
{
    for_all_time_slices
    {
        // if there is no load then there are no channels and no runs
        if( TotalLoad( link_loads[time_slice], EDirection(direction) ) == 0 )
        {
            for_both_near_and_far
            {
                num_runs[direction][time_slice][near_far]        = 0.0;
                num_channels[direction][time_slice][near_far]    = 0.0;
            }
        }

        // if there is some load work out channels
        // and runs for both directions
        if( TotalLoad( link_loads[time_slice], EDirection(direction) ) > 0 )
        {
            // calculate the total runs and total channels
            float total_runs
            = max( k_minimum_number_of_runs, ceil(PeakLoad(link_loads[time_slice],
                   links_spans, links_far_all_stops_pos[direction])/k_bus_capacity) );

            float total_channels = min( k_maximum_channels,
                                        floor(total_runs/k_minimum_number_of_runs) );

            // calculate the maximum span load to the near and far stops
            float maximum_span_load[ENearFarSize];
            maximum_span_load[eNear] = 0;
            maximum_span_load[eFar]  = 0;
            for_all_spans
            {
                ENearFar near_far = ( span < k_outermost_stop_near ? eNear : eFar );

                maximum_span_load[near_far]
                = max( maximum_span_load[near_far], SpanLoad( link_loads[time_slice],
                  links_spans, links_far_all_stops_pos[direction], span ) );
            }

            // calculate the excess on the near part on the line
            float excess_near = (maximum_span_load[eNear]-maximum_span_load[eFar])/
                                (maximum_span_load[eNear]+maximum_span_load[eFar]);
```

```
            //cout << direction << space << time_slice << space << total_runs << space
            //     << total_channels << space
            //     << TotalLoad( link_loads[time_slice], EDirection(direction) ) << endl;

            // is there enough load to activate the near channel
            bool near_channel_active
            = floor(total_runs*excess_near) >= k_minimum_number_of_runs &&
              ( total_runs - floor(total_runs*excess_near) >= k_minimum_number_of_runs );

            // runs for near/far stop
            num_runs[direction][time_slice][eNear]
            = ( near_channel_active ? floor(total_runs*excess_near) : 0.0 );

            num_runs[direction][time_slice][eFar]
            = total_runs - num_runs[direction][time_slice][eNear];

            // channels for near/far stop
            num_channels[direction][time_slice][eNear]
            = ( near_channel_active ? max( float(1.0), floor(((num_runs[direction]
              [time_slice][eNear]/total_runs)*total_channels)) ) : 0.0 );

            num_channels[direction][time_slice][eFar]
            = total_channels - num_channels[direction][time_slice][eNear];

            // manual override
            if( direction == eInbound )
            {
                if( time_slice == 12 ) num_channels[direction][time_slice][eFar] = 3.0;
                if( time_slice == 21 ) num_channels[direction][time_slice][eFar] = 2.0;
            }
            if( direction == eOutbound )
            {
                if( time_slice == 19 ) num_channels[direction][time_slice][eNear] = 2.0;
                if( time_slice == 19 ) num_channels[direction][time_slice][eFar]  = 1.0;

                if( time_slice == 21 ) num_channels[direction][time_slice][eFar]  = 1.0;

                if( time_slice == 62 ) num_channels[direction][time_slice][eNear] = 0.0;
                if( time_slice == 62 ) num_runs[direction][time_slice][eNear]     = 0.0;
                if( time_slice == 62 ) num_channels[direction][time_slice][eFar]  = 3.0;
                if( time_slice == 62 ) num_runs[direction][time_slice][eFar]      = 16.0;

                if( time_slice == 63 ) num_channels[direction][time_slice][eNear] = 0.0;
                if( time_slice == 63 ) num_runs[direction][time_slice][eNear]     = 0.0;
                if( time_slice == 63 ) num_channels[direction][time_slice][eFar]  = 3.0;
                if( time_slice == 63 ) num_runs[direction][time_slice][eFar]      = 15.0;
            }
        }
    }
}

// ***
// *** convert channel data into easier to work with format
// ***

bool        channel_is[k_channels_size][EDirectionSize][k_time_slices_size];
ENearFar    channel_type[k_channels_size][EDirectionSize][k_time_slices_size];

for_all_channels
{
    for_both_directions
    {
        for_all_time_slices
        {
            channel_is[channel][direction][time_slice]      = false;
            channel_type[channel][direction][time_slice]    = eFar;
        }
    }
}
```

```
for_both_directions
{
    for_all_time_slices
    {
        // far
        for( int channel = 0;
             channel < num_channels[direction][time_slice][eFar];
             channel++ )
        {
            channel_is[channel][direction][time_slice]      = true;
            channel_type[channel][direction][time_slice]    = eFar;
        }

        // near
        for( int channel = num_channels[direction][time_slice][eFar];
             channel < num_channels[direction][time_slice][eFar] +
             num_channels[direction][time_slice][eNear];
             channel++ )
        {
            channel_is[channel][direction][time_slice]      = true;
            channel_type[channel][direction][time_slice]    = eNear;
        }
    }
}

// ***
// *** exhaust express options
// ***

// for output data
ostringstream text_feedback[EDirectionSize][k_time_slices_size];

// for metric assessment
vector< pair<long,double> >   metric_data[k_channels_size][EDirectionSize]
                                                          [k_time_slices_size];

for_all_express_channels_reverse
{
    // print whether near or far channel
    for_both_directions
        for_all_time_slices
            if( channel_is[channel][direction][time_slice] )
                text_feedback[direction][time_slice]
                << "channel "
                << ENearFarName(channel_type[channel][direction][time_slice])
                << endl;

    // assess all-stop run group option
    float all_stop_run_group_number_of_runs[EDirectionSize][k_time_slices_size];
    for_both_directions
        for_all_time_slices
            if( channel_is[channel][direction][time_slice] )
                all_stop_run_group_number_of_runs[direction][time_slice]
                = max( k_minimum_number_of_runs,
                       ceil(PeakLoad(link_loads[time_slice],
                       links_spans, links_far_all_stops_pos[direction])/k_bus_capacity) );

    // declare the express run group control variables
    bool        best_express_run_group_found[EDirectionSize][k_time_slices_size];
    vector<int> best_express_run_group_stops[EDirectionSize][k_time_slices_size];
    vector<int> best_express_run_group_links[EDirectionSize][k_time_slices_size];
    float       best_express_run_group_number_of_runs[EDirectionSize][k_time_slices_size];
    float       best_express_run_group_metric[EDirectionSize][k_time_slices_size];

    // set the default values for the express group control variables
    for_both_directions
    {
        for_all_time_slices
        {
            if( channel_is[channel][direction][time_slice] )
            {
                best_express_run_group_found[direction][time_slice]  = false;
                best_express_run_group_metric[direction][time_slice]
                = all_stop_run_group_number_of_runs[direction]
                  [time_slice]*float(k_intermediate_stops_size);
            }
        }
    }
```

```
// ***
// *** find the best express run group
// ***

// make all intermediate stops (needed by combination algorithm)
vector<int> all_intermediate_stops[ENearFarSize];
for_both_near_and_far
    for( int stop = (k_turn_around_stop[near_far]-1);
         stop > Line().InnermostStop(); stop-- )
    {
        all_intermediate_stops[near_far].push_back( stop );
    }

// for each combination stop level (stop at 1 intermediate stop,
// stop at 2 intermediate stops, etc)
int combination = 0;
for( int combination_stop_level = 0;
     combination_stop_level <= (k_outermost_stop_far-1);
     combination_stop_level++ )
{
    // initialise intermediate stops (needed by combination algorithm)
    vector<int> combination_intermediate_stops[ENearFarSize];
    for_both_near_and_far
        if( !( near_far == eNear && combination_stop_level >=
               k_outermost_stop_near ) )
            for( int stop = 0; stop < combination_stop_level; stop++ )
                combination_intermediate_stops[near_far]
                .push_back( k_turn_around_stop[near_far]-1-stop );

    // cycle through all combinations
    do
    {
        // viusal free-back for slow progress
        if( IsWhole(float(combination)/1000.0) )
            cout << combination << space
                 << round((float(combination)/float(k_combs_size))*100.0)
                 << "%" << endl;

        // make the stops for this combination
        vector<int> combination_all_stops[ENearFarSize];
        for_both_near_and_far
        {
            copy_all( combination_intermediate_stops[near_far],
                      combination_all_stops[near_far] );
            combination_all_stops[near_far].push_back( k_turn_around_stop[near_far] );
            combination_all_stops[near_far].push_back( k_innermost_stop );
        }

        // make the positive and negative links for this combination
        // positive links are used by the express run group
        // neagtive links (i.e. the remainer) are used by the all-stop run group
        vector<int> combination_links[ENearFarSize][EDirectionSize][EPosNegSize];
        for_all_stops_from
        {
            for_all_stops_to
            {
                if( from != to )
                {
                    for_both_near_and_far
                    {
                        EPosNeg pos_neg
                        = ( contains(combination_all_stops[near_far],from) &&
                            contains(combination_all_stops[near_far],to) ?
                            ePos :
                            eNeg );
                        combination_links[near_far][Direction(from,to)][pos_neg]
                        .push_back(Link(from,to));
                    }
                }
            }
        }
```

```
// see if this combination is better than the current best one
for_both_directions
{
    for_all_time_slices
    {
        if( channel_is[channel][direction][time_slice] )
        {
            // find the express run group number of runs
            float express_run_group_number_of_runs
            = max( k_minimum_number_of_runs,
              ceil(PeakLoad(link_loads[time_slice], links_spans,
              combination_links[channel_type[channel][direction][time_slice]]
              [direction][ePos])/k_bus_capacity) );

            // find the number of runs for this channel
            float runs_for_this_channel
             = num_runs[direction][time_slice]
                       [channel_type[channel][direction][time_slice]] /
              num_channels[direction][time_slice]
                           [channel_type[channel][direction][time_slice]];

            // only after express run groups with a certain number of runs
            if( (express_run_group_number_of_runs ==
                 floor(runs_for_this_channel)) ||
                (express_run_group_number_of_runs ==
                 ceil(runs_for_this_channel)) )
            {
                // find the all-stop run group number of runs
                // minus the express run group number of runs
                float all_stop_run_group_number_of_runs
                      _when_express_run_group_removed
                = max( k_minimum_number_of_runs,
                       ceil(PeakLoad(link_loads[time_slice], links_spans,
                       combination_links[channel_type[channel]
                       [direction][time_slice]][direction][eNeg])
                       /k_bus_capacity) );

                float express_run_group_plus_all_stop_run_group_metric
                = express_run_group_number_of_runs *
                  float(combination_stop_level) +
                  all_stop_run_group_number_of_runs
                  _when_express_run_group_removed *
                  float(k_intermediate_stops_size);

                if( express_run_group_plus_all_stop_run_group_metric <
                    best_express_run_group_metric[direction][time_slice] )
                {
                    // update best_run_group data
                    best_express_run_group_found[direction][time_slice]
                    = true;

                    best_express_run_group_metric[direction][time_slice]
                    = express_run_group_plus_all_stop_run_group_metric;

                    best_express_run_group_number_of_runs
                    [direction][time_slice]
                    = express_run_group_number_of_runs;

                    erase_all( best_express_run_group_links[direction]
                                                           [time_slice] );

                    copy_all( combination_links[channel_type[channel]
                              [direction][time_slice]][direction][ePos],
                              best_express_run_group_links
                              [direction][time_slice] );

                    erase_all( best_express_run_group_stops[direction]
                                                           [time_slice] );
                    copy_all( combination_all_stops
                              [channel_type[channel][direction][time_slice]],
                              best_express_run_group_stops
                              [direction][time_slice] );
```

```
                                // add data for metric assessment
                                if( channel_type[channel][direction][time_slice] == eFar )
                                    metric_data[channel][direction][time_slice].push_back
                                    ( pair<long,double>( combination,
                                    express_run_group_plus_all_stop_run_group_metric ) );
                            }
                        }
                    }
                }
            }

            combination++;

            // advance near combination
            if( combination_stop_level != 0 )
                next_combination_all( all_intermediate_stops[eNear],
                                      combination_intermediate_stops[eNear] );
        }
        while( combination_stop_level != 0 &&
               next_combination_all( all_intermediate_stops[eFar],
                                     combination_intermediate_stops[eFar] ) );
    }

    // ***
    // *** run the best express run group
    // ***

    for_both_directions
    {
        for_all_time_slices
        {
            if( channel_is[channel][direction][time_slice] )
            {
                if( best_express_run_group_found[direction][time_slice] )
                {
                    // make an express run group
                    make_one_run_group(
                    time_slice, EDirection(direction),
                    best_express_run_group_number_of_runs[direction][time_slice],
                    best_express_run_group_stops[direction][time_slice],
                    best_express_run_group_links[direction][time_slice],
                    link_loads[time_slice], links_spans, runs,
                    text_feedback[direction][time_slice] );

                    // update channel guides
                    num_channels[direction][time_slice]
                    [channel_type[channel][direction][time_slice]]--;

                    num_runs[direction][time_slice]
                    [channel_type[channel][direction][time_slice]] -=
                    best_express_run_group_number_of_runs[direction][time_slice];
                }
                else
                {
                    num_channels[direction][time_slice]
                    [channel_type[channel][direction][time_slice]]--;

                    text_feedback[direction][time_slice]
                    << "cancelled channel" << endl << endl;
                }
            }
        }
    }
}
```

```
    // ***
    // *** once express options exhausted run all-stop
    // ***

    for_both_directions
    {
        for_all_time_slices
        {
            // channel 0 is the all-stop channel by-definition
            if( channel_is[0][direction][time_slice] )
            {
                // when the express channels are exhausted do single all-stop channel
                float number_of_runs = num_runs[direction][time_slice][eFar];

                make_one_run_group( time_slice, EDirection(direction),
                                    number_of_runs, stops_far_all_stops,
                                    links_far_all_stops_pos[direction],
                                    link_loads[time_slice], links_spans, runs,
                                    text_feedback[direction][time_slice] );

                // update channel guides
                num_channels[direction][time_slice][eFar]--;
                num_runs[direction][time_slice][eFar] = 0;
            }
        }
    }

    // check all channels and runs are zeroed
    for_both_directions
    {
        for_all_time_slices
        {
            for_both_near_and_far
            {
                assert( num_channels[direction][time_slice][near_far] == 0 );
                assert( num_runs[direction][time_slice][near_far]     == 0 );
            }
        }
    }

    // print feedback
    for_both_directions
    {
        for_all_time_slices
        {
            cout << time_slice << space << EDirectionName(direction) << endl << endl;

            cout << text_feedback[direction][time_slice].str();
        }
    }

    for_both_directions
        for_all_time_slices
            for_all_channels_reverse
            {
                for_all( vector<  pair_long_double_T >,
                         metric_data[channel][direction][time_slice], data )
                {
                    if( metric_data[channel][direction][time_slice].size() >= 4 )
                        clog << channel                          << " "
                             << " " << EDirectionName(direction) << " "
                             << TimeSliceDuration(time_slice)    << " "
                             << data->first                      << " "
                             << log10(data->first+1)             << " "
                             << data->second                     << endl;
                }

                if(metric_data[channel][direction][time_slice].size() >= 4 )
                    clog << endl;
            }

    WriteRuns( runs );

    WriteFile( "oss_log.txt", oss_log.str() );
}
```

```
//
// DeadheadRuns
//

struct less_than_outermost_FO
{
    bool operator() (const cRun& a, const cRun& b)
    {
        bool add_slack_time = true;

        return a.Time(a.Stop(eOuter),add_slack_time) < b.Time(b.Stop(eOuter),add_slack_time);
    }
};

void
do_pair( vector<cRun>& iRuns, EInnerOuter iInnerOuter, cInstant iAllowedHoldTime )
{
    vector<cRun> paired_runs;

    // sort on innermost or outermost stop
    if( iInnerOuter == eInner )
        sort_all( iRuns );
    else
        sort( all(iRuns), less_than_outermost_FO() );

    // cycle through until no more pairs
    bool found_pair;
    do
    {
        found_pair = false;

        for_all( vector<cRun>, iRuns, run_current )
        {
            vector<cRun>:: iterator run_next = run_current; run_next++;

            if( run_next != iRuns.end() )
            {
                // determine if the pairs are pairable
                bool pairable = false;
                if( iInnerOuter == eInner &&
                    ( run_current->IsDirection(eInbound) &&
                      run_next->IsDirection(eOutbound) ) )
                {
                    pairable = true;
                }

                if( iInnerOuter == eOuter &&
                    ( run_current->IsDirection(eOutbound) &&
                      run_next->IsDirection(eInbound) ) )
                {
                    pairable = true;
                }

                if( pairable )
                {
                    // if within time overlap
                    bool add_slack_time = true;

                    if( cDuration( run_current->Time(iInnerOuter==eInner?
                                   k_innermost_stop:run_current->Stop(eOuter),add_slack_time),
                                   run_next->Time(iInnerOuter==eInner?
                                   k_innermost_stop:run_next->Stop(eOuter),   add_slack_time)
                                 )
                        .Width() < iAllowedHoldTime )
                    {
                        run_current->MarkAsPaired();
                        run_next->MarkAsPaired();
                        found_pair = true;
                    }
                }
            }
        }
```

```
        // shift out paired runs
        vector<cRun> temp;
        for_all( vector<cRun>, iRuns, run )
            if( run->IsPaired() )
                paired_runs.push_back(*run);
            else
                temp.push_back(*run);
        erase_all(iRuns);
        copy_all(temp,iRuns);

        // re-sort
        if( iInnerOuter == eInner )
            sort_all( iRuns );
        else
            sort( all(iRuns), less_than_outermost_FO() );
    }
    while( found_pair );

    // recombine paired with unpaired
    copy_all( paired_runs, iRuns );
}

void
DeadheadRuns()
{
    vector<cRun> runs = ReadRuns();
    cout << "total runs before deadheading: " << runs.size() << endl;

    // pair on the innermost stop where possible.
    do_pair( runs, eInner, cRun( eInbound, k_zero_seconds, k_innermost_stop,
                                 k_outermost_stop_far ).Duration().Width() );

    // deadhead unpaired runs between innermost and the near outermost stop
    vector<cRun> deadhead_runs;
    cInstant time_to_centre = cRun( eInbound, k_zero_seconds, k_innermost_stop,
                                    k_outermost_stop_near ).Time(k_innermost_stop);

    for_all( vector<cRun>, runs, run )
    {
        if( !run->IsPaired() )
        {
            // the direction the deadhead run is the opposite of the service run
            EDirection deadhead_dir = (run->IsDirection(eInbound)?eOutbound:eInbound);

            // outbound deadhead buses leave the innermost stop as the service run arrives
            // inbound deadhead buses arrive at the innermost stop as the service run leaves
            cInstant deadhead_centre_time
            = cInstant(run->Time(k_innermost_stop)) +
              ( deadhead_dir==eOutbound ? time_to_centre : -time_to_centre );

            // construct the deadhead run
            cRun deadhead_run = cRun( deadhead_dir, deadhead_centre_time,
                                      k_innermost_stop, k_outermost_stop_near );

            // store the deadhead run
            deadhead_runs.push_back( deadhead_run );
        }
    }

    copy_all( deadhead_runs, runs );
    cout << "total runs after deadheading:  " << runs.size() << endl << endl;

    // clear previous pairing
    for_all( vector<cRun>, runs, run )
        run->MarkAsUnpaired();

    vector<cRun> unpaired_runs, completed_runs;

    // far  outermost stop to innermost stop runs are now done
    // near outermost stop to innermost stop runs are not yet done
    for_all( vector<cRun>, runs, run )
    {
        if( run->Stop(eOuter) == k_outermost_stop_far )
            completed_runs.push_back( *run );
        else
            unpaired_runs.push_back( *run );
    }
```

```
    // pair on the near outermost stop where possible.
    do_pair( unpaired_runs, eOuter,
             cInstant( cRun( eInbound, k_zero_seconds, k_innermost_stop,
                             k_outermost_stop_near ).Duration().Width().AsSeconds() * 2.0 )
             + k_run_slack );

    // extend those runs that remain unpaired back to the outermost stop
    for_all( vector<cRun>, unpaired_runs, unpaired_run )
    {
        if( !unpaired_run->IsPaired() )
        {
            if( unpaired_run->PaxCount()==0 )
            {
                unpaired_run->EraseAllStops();
                unpaired_run->AddStop(k_innermost_stop);
                unpaired_run->AddStop(k_outermost_stop_far);
            }
            else
                unpaired_run->AddStop(k_outermost_stop_far);
        }
    }

    // now completed
    copy_all( unpaired_runs, completed_runs );

    // write out
    WriteRuns( completed_runs );
}

//
// MakePaxGroupsFromRuns
//

void
MakePaxGroupsFromRuns()
{
    vector<cRun> runs = ReadRuns();

    vector<cPaxGroup> pax_groups;
    for_all( vector<cRun>, runs, run )
        run->MakePaxs( pax_groups );

    WritePaxGroups( pax_groups );
}

//
// MakePaxsFromLoads
//

void
MakePaxGroupsFromLoads()
{
    const cInstant k_average_wait_time = cInstant("00:01:00");
    const cInstant k_line_travel_time  = cInstant("00:30:00");

    // get loads
    cout << "get loads" << endl;
    float from_to_loads[k_time_slices_size][k_stops_size][k_stops_size];
    ReadLoads( from_to_loads, "loads.txt" );

    // convert from-stop/to-stop loads to link loads
    cout << "convert from-stop/to-stop loads to link loads" << endl;
    float link_loads[k_time_slices_size][k_links_size];
    for_all_time_slices
        for_all_stops_from
            for_all_stops_to
                link_loads[time_slice][Link(from,to)] = from_to_loads[time_slice][from][to];
```

```
    vector<cPaxGroup> pax_groups;
    for_all_time_slices
    {
        cout << time_slice << endl;

        for_all_links
        {
            if( link_loads[time_slice][link] != 0 )
            {
                pax_groups.push_back (
                cPaxGroup( link_loads[time_slice][link],
                           cInstant(TimeSliceDuration(time_slice).Centre() -
                           k_average_wait_time),
                           cLink(link),
                           TimeSliceDuration(time_slice).Centre(),
                           cInstant(TimeSliceDuration(time_slice).Centre() +
                           ((cLink(link).Distance()/Line().LineLength())*k_line_travel_time))
                         ) );
            }
        }
    }

    WritePaxGroups( pax_groups );
}

//
// PaxGroupAnalysis
//

void
PaxGroupAnalysis()
{
    vector<cPaxGroup> pax_groups = ReadPaxGroups();
    AnalysePaxs( pax_groups, eInboundExtended );
    AnalysePaxs( pax_groups, eOutboundExtended );
    AnalysePaxs( pax_groups, eEitherDirection );
}

//
// RunAnalysis
//

void
RunAnalysis()
{
    vector<cRun> runs = ReadRuns();

    //
    // calculate average run time
    //

    double total_run_time = 0, total_run_dist = 0;
    for_all( vector<cRun>, runs, run )
    {
        if( run->PaxCount() > 0 &&
            run->Duration().Start() > cInstant("05:00:00") &&
            run->Duration().End() < cInstant("25:00:00") )
        {
            total_run_time += run->Duration().Width().AsSeconds();
            total_run_dist += run->Distance();
        }
    }
    clog << "average: " << total_run_dist/total_run_time << endl;
```

```
    // ***
    // *** calcualte the peak number of runs and the run drain profile
    // ***

    // build time map
    bool add_slack_time = true;
    vector<pair_cinstant_int_t> run_counters;
    for_all( vector<cRun>, runs, run )
    {
        if(  run->Stop(eOuter)==17 )
        {
            if( run->IsDirection(eInbound) )
            {
                run_counters.push_back(
                pair<cInstant,int>(run->Time(run->Stop(eOuter) ),-1) );
            }
            else
            {
                run_counters.push_back(
                pair<cInstant,int>(run->Time(run->Stop(eOuter),add_slack_time) ,1) );
            }
        }
    }

    // time slice run drain level
    int run_drain[k_time_slices_size];
    for_all_time_slices
        run_drain[time_slice] = 0;

    // go through run events
    int run_level = 0, run_maximum = 0;
    cout << "line open: " << space << 0 << endl;
    sort( all(run_counters), pair_cinstant_int_t_FO() );
    for_all( vector<pair_cinstant_int_t>, run_counters, run_counter )
    {
        // current and maximum level
        run_level += run_counter->second;
        run_maximum = max( run_maximum, -run_level );
        cout << run_counter->first.RoundToSeconds() << space << -run_level << endl;

        // time slice level
        int index = floor(run_counter->first/k_time_slice_length) - k_time_slice_offset;
        run_drain[index] = max( -run_level, run_drain[index] );
    }
    cout << endl << "maximum: " << run_maximum << endl << endl;

    // make run drain file
    ostringstream oss_drain;
    oss_drain << endl;
    for_all_time_slices
        oss_drain << run_drain[time_slice] << endl;
    WriteFile( "runs drain.txt", oss_drain.str() );

    // ***
    // *** do general analysis of the runs
    // ***

    AnalyseRuns( runs, eInboundExtended );
    AnalyseRuns( runs, eOutboundExtended );
    AnalyseRuns( runs, eEitherDirection );
}
```

```
//
// CalculateLineRatio
//

void
CalculateLineRatio()
{
    // get loads
    cout << "get loads" << endl;
    float from_to_loads[k_time_slices_size][k_stops_size][k_stops_size];
    ReadLoads( from_to_loads, "loads.txt" );

    cLink links[15];
    links[0 ] = cLink( 17, 16 );
    links[1 ] = cLink( 17, 13 );
    links[2 ] = cLink( 17,  5 );
    links[3 ] = cLink( 17,  2 );
    links[4 ] = cLink( 17,  0 );
    links[5 ] = cLink( 16, 13 );
    links[6 ] = cLink( 16,  5 );
    links[7 ] = cLink( 16,  2 );
    links[8 ] = cLink( 16,  0 );
    links[9 ] = cLink( 13,  5 );
    links[10] = cLink( 13,  2 );
    links[11] = cLink( 13,  0 );
    links[12] = cLink(  5,  2 );
    links[13] = cLink(  5,  0 );
    links[14] = cLink(  2,  0 );

    float loads[15];
    for_all_time_slices
    {
        for( int i = 0; i < 15; i++ )
        {
            if( time_slice == 0 )
                loads[i] = 0;

            loads[i] += from_to_loads[time_slice][links[i].From()][links[i].To()  ];
            loads[i] += from_to_loads[time_slice][links[i].To()  ][links[i].From()];
        }
    }

    float total_load = 0;
    for( int i = 0; i < 15; i++ )
        total_load += loads[i];

    for( int i = 0; i < 15; i++ )
    {
        loads[i] = loads[i]/total_load;
        cout << loads[i] << endl;
    }

    float total_metres = 0;
    for( int i = 0; i < 15; i++ )
        total_metres += loads[i] * links[i].Distance();

    cout << "ratio:" << ( total_metres/13296.0 ) * 100.0 << endl;
}
```

```
//
// main
//

void print_menu()
{
    cout << "menu options ( make a choice and hit return )" << endl << endl;

    cout << "a - calculate line ratio" << endl;
    cout << "b - convert transmilenio run files to matilda run files" << endl;
    cout << "c - convert transmilenio load files to matilda load files" << endl;
    cout << "d - make run groups from loads" << endl;
    cout << "e - deadhead runs" << endl;
    cout << "f - make pax groups from run groups" << endl;
    cout << "g - make pax groups from loads" << endl;
    cout << "h - produce pax group analysis data" << endl;
    cout << "i - produce run group analysis data" << endl;
    cout << "q - quit" << endl << endl;
}

int main()
{
    try
    {
        cout << "matilda - ultra-efficient brt timetabling" << endl << endl;

        char menu_choice;

        print_menu();
        cout << "menu choice: ";
        cin >> menu_choice;
        cout << endl;

        while ( menu_choice != 'q' )
        {
            if( menu_choice == 'a' )
                CalculateLineRatio();

            else if( menu_choice == 'b' )
                ConvertRunData();

            else if( menu_choice == 'c' )
            {
                ConvertLoadData();
                MakeLoads();
            }

            else if( menu_choice == 'd' )
                MakeRunGroups();

            else if( menu_choice == 'e' )
                DeadheadRuns();

            else if( menu_choice == 'f' )
                MakePaxGroupsFromRuns();

            else if( menu_choice == 'g' )
                MakePaxGroupsFromLoads();

            else if( menu_choice == 'h' )
                PaxGroupAnalysis();

            else if( menu_choice == 'i' )
                RunAnalysis();

            else if( menu_choice == 'q' )
                break;

            else
                cout << "error : unknown menu choice" << endl << endl;

            cout << endl;
            print_menu();
            cout << "menu choice: ";
            cin >> menu_choice;
            cout << endl;
        }
    }
```

```
    catch( string error )
    {
        cout << "an internal program error has occured" << endl;
        cout << "error reference: " << error << endl;
        cout << "program exited unexpectedly" << endl;
        return 0;
    }

    catch(...)
    {
        cout << "an internal program error has occured" << endl;
        cout << "program exited unexpectedly" << endl;
        return 0;
    }

    cout << "program exited normally" << endl;
    return 0;
}
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * convert.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * convert.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * convert.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */


#pragma once

void    ConvertRunData();
void    ConvertLoadData();
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * convert.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * convert.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * convert.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "convert.h"

#include "load.h"
#include "run.h"

#include <iostream>
#include <algorithm>
#include <cmath>
using namespace std;

//
// ConvertRunData
//

int
convert_to_stop( int iTransmilenioPlatform )
{
    // stops
    if( iTransmilenioPlatform == 951 ||
        iTransmilenioPlatform == 952 ||
         iTransmilenioPlatform == 953 )
    {
        return 17;
    }
    if( iTransmilenioPlatform == 4072 )
        return 13;
    if( iTransmilenioPlatform == 7026 || iTransmilenioPlatform == 8026 )
        return 0;

    // outside stops
    if( iTransmilenioPlatform == 926 || iTransmilenioPlatform == 904 )
        return -1;

    // depots
    if( iTransmilenioPlatform == 5 )
        return -2;

    // outside depots
    if( iTransmilenioPlatform == 3 || iTransmilenioPlatform == 1 )
        return -3;

    throw string ( "unknown transmilenio platform" );

    // happy compiler
    return 0;
}

bool
is_stop( int iStop )
{
    return iStop >= -1;
}

bool
is_inside_stop( int iStop )
{
    return iStop >= 0;
}
```

```
bool
is_outside_stop( int iStop )
{
    return iStop == -1;
}

bool
is_depot( int iStop )
{
    return iStop == -2 || iStop == -3;
}

bool
is_inside_depot( int iStop )
{
    return iStop == -2;
}

bool
is_outside_depot( int iStop )
{
    return iStop == -3;
}

bool
is_outside( int iStop )
{
    return is_outside_stop(iStop) || is_outside_depot(iStop);
}

bool
is_inside( int iStop )
{
    return !is_outside(iStop);
}

bool
is_both_outside( int iFrom, int iTo )
{
    return is_outside(iFrom) && is_outside(iTo);
}

bool
is_origin( int iStop )
{
    return iStop == 17 || iStop == 13;
}

void
ConvertRunData()
{
    ostringstream oss_log;

    string filename[4];
    filename[0] = "transmilenio run 12.txt";
    filename[1] = "transmilenio run 16.txt";
    filename[2] = "transmilenio run 18.txt";
    filename[3] = "transmilenio run 19.txt";

    int num_buses[4];
    num_buses[0] = 62;
    num_buses[1] = 42;
    num_buses[2] = 28;
    num_buses[3] = 29;

    vector<cRun> runs;

    for( int file = 0; file < 4; file++ )
    {
        vector<pair_cinstant_int_t> run_counters;

        oss_log << "file" << space << file << endl;

        vector<string> file_lines = GetLines( ReadFile(filename[file]) );

        int    bus_count = 0;
```

```
// sort events by bus
vector<string>    bus_events[100];
for( int line = 0; line < file_lines.size(); line++ )
{
    vector<string> cells = GetCells( file_lines[line] );
    bus_count = max( ToInt(cells[2]), bus_count );
    bus_events[ToInt(cells[2])-1].push_back( file_lines[line] );
}

// group events into runs
vector<string>    bus_runs[100];
for( int bus = 0; bus < bus_count; bus++ )
{
    for_all( vector<string>, bus_events[bus], line )
    {
        vector<string>::iterator next = line; next++;

        if( next != bus_events[bus].end() )
        {
            bus_runs[bus].push_back( *line + string("\t") + *next );
        }
    }
}

// for each bus
for( int bus = 0; bus < bus_count; bus++ )
{
    oss_log << "bus " << bus << endl;

    // for each run
    for_all( vector<string>, bus_runs[bus], line )
    {
        vector<string> cells = GetCells( *line );

        cInstant    from_time    = cInstant(cells[0]),
                    to_time      = cInstant(cells[4]);

        long        from_stop    = convert_to_stop(ToInt(cells[3])),
                    to_stop      = convert_to_stop(ToInt(cells[7]));

        // if not the same stop or a trip between a depot and an origin stop
        if( !(from_stop == to_stop)                          &&
            !(is_depot(from_stop)  && is_origin(to_stop))    &&
            !(is_origin(from_stop) && is_depot(to_stop))     )
        {
            // get run length
            cInstant span = cDuration(from_time, to_time).Width();

            // adjust runs that start/stop outside of the line
            if( is_outside_stop(from_stop) && is_inside_stop(to_stop) )
            {
                if( file == 0 && span > cInstant("00:50:00") )
                    from_time = to_time - cInstant("00:21:00");
                if( file == 0 && span < cInstant("00:50:00") )
                    from_time = to_time - cInstant("00:19:00");

                if( file == 1 && span > cInstant("01:00:00") )
                    from_time = to_time - cInstant("00:28:00");
                if( file == 1 && span < cInstant("01:00:00") )
                    from_time = to_time - cInstant("00:26:00");

                from_stop = 0;
            }

            // adjust runs that start/stop outside of the line
            if( is_outside_stop(to_stop) && is_inside_stop(from_stop) )
            {
                if( file == 0 && span > cInstant("00:50:00") )
                    to_time = from_time + cInstant("00:21:00");
                if( file == 0 && span < cInstant("00:50:00") )
                    to_time = from_time + cInstant("00:19:00");

                if( file == 1 && span > cInstant("01:00:00") )
                    to_time = from_time + cInstant("00:28:00");
                if( file == 1 && span < cInstant("01:00:00") )
                    to_time = from_time + cInstant("00:26:00");

                to_stop = 0;
            }
```

```
// outermost stop far becomes the depot for all lines
// except file 0 which is outermost stop near
if( is_depot(from_stop) )
{
    if( to_stop == k_innermost_stop || is_outside_stop(to_stop) )
    {
        if( file != 0 )
        {
            from_time = to_time - cInstant("00:23:00");
            from_stop = k_outermost_stop_far;
        }
        else
        {
            from_time = to_time - cInstant("00:15:00");
            from_stop = k_outermost_stop_near;
        }
    }

    if( is_outside_stop(to_stop) )
        to_stop = k_innermost_stop;
}

// outermost stop far becomes the depot for all lines
// except file 0 which is outermost stop near
if( is_depot(to_stop) )
{
    if( from_stop == k_innermost_stop || is_outside_stop(from_stop) )
    {
        if( file != 0 )
        {
            to_time = from_time + cInstant("00:23:00");
            to_stop = k_outermost_stop_far;
        }
        else
        {
            to_time = from_time + cInstant("00:15:00");
            to_stop = k_outermost_stop_near;
        }
    }

    if( is_outside_stop(from_stop) )
        from_stop = k_innermost_stop;
}

oss_log << from_time << tab
        << to_time   << tab
        << from_stop << tab
        << to_stop   << tab
        << *line     << endl;

// just to feed to run constructor
vector<int> temp_stops;
temp_stops.push_back(from_stop);
temp_stops.push_back(to_stop);

// construct run
runs.push_back( cRun( cLink(from_stop,to_stop).Direction(),
                      from_time, to_time, temp_stops ) );

// from stop outermost
if( (file == 0 && from_stop == 13) || from_stop == 17 )
{
    run_counters.push_back( pair<cInstant,int>(from_time,-1) );
}

// to stop outermost
if( (file == 0 && to_stop == 13) || to_stop == 17 )
{
    run_counters.push_back( pair<cInstant,int>(to_time,1) );
}

// from stop innermost
if( (file == 0 || file == 1) && from_stop == 0 )
{
    run_counters.push_back( pair<cInstant,int>(from_time,-1) );
}
```

```
                    // to stop innermost
                    if( (file == 0 || file == 1) && to_stop == 0 )
                    {
                        run_counters.push_back( pair<cInstant,int>(to_time,1) );
                    }
                }
            }
        }

        // time slice run drain level
        int run_drain[k_time_slices_size];
        for_all_time_slices
            run_drain[time_slice] = 0;

        // go through run events
        int run_level = 0, run_maximum = 0;
        cout << k_open_time << space << 0 << endl;
        sort( all(run_counters), pair_cinstant_int_t_FO() );
        for_all( vector<pair_cinstant_int_t>, run_counters, run_counter )
        {
            // current and maximum level
            run_level += run_counter->second;
            run_maximum = max( run_maximum, -run_level );
            cout << run_counter->first.RoundToSeconds() << space << -run_level << endl;

            // time slice level
            int index = floor(run_counter->first/k_time_slice_length) - k_time_slice_offset;
            run_drain[index] = max( -run_level, run_drain[index] );
        }
        cout << endl << "maximum: " << run_maximum << endl << endl;

        // make run drain file
        ostringstream oss_runs;
        oss_runs << endl;
        for_all_time_slices
            oss_runs << run_drain[time_slice] << endl;
        WriteFile( "runs_drain " + filename[file], oss_runs.str() );
    }

    WriteRuns( runs );
    WriteFile( "oss_log.txt", oss_log.str() );
}

//
// ConvertLoadData
//

void
ConvertLoadData()
{
    ostringstream oss_log;

    // all external stops have been made to be stop 0
    float entry_exit_ratio[EEntryExitSize];
    entry_exit_ratio[eEntry] = 17137.0/219242.0; // probability that if you enter stop 0
                                                 // you will exit on the line
    entry_exit_ratio[eExit]  = 22920.0/225025.0; // probability if you exit stop 0
                                                 // that you have entered on the line

    for_all_days
    {
        cout << EDayName(day) << endl;

        // zero container
        float loading[k_time_slices_size][k_stops_size][EEntryExitSize];
        for_all_time_slices
            for_all_stops
                for( int entry_exit = 0; entry_exit < EEntryExitSize; entry_exit++ )
                    loading[time_slice][stop][entry_exit] = 0;

        // get this days loads
        vector<string> file_lines = GetLines( ReadFile(string("transmilenio loads ") +
                                                       EDayName(day) + string(".txt")) );
```

```
// process lines
const_for_all( vector<string>, file_lines, file_line )
{
    //cout << *file_line << endl;

    // get cells
    vector<string> data_cells = GetCells( *file_line );
    if( data_cells.size() != 5 )
        throw string( "cells not equal to five" );

    // get entry/exit
    if( !(data_cells[3]=="E" || data_cells[3]=="S") )
        throw string( "unknown case" );
    EEntryExit entry_exit = ( data_cells[3]=="E" ? eEntry : eExit );

    int stop_index = 0;
    if( Line().IsStopExternalIndex(ToInt(data_cells[0]))
        && ToInt(data_cells[0]) != 9110 )
    {
        stop_index = Line().ExternalIndexToStop(ToInt(data_cells[0]));
    }

    // get load
    float load = ToFloat(data_cells[4]);
    if( stop_index == 0 )
        load *= entry_exit_ratio[entry_exit];

    // get time index
    vector<string> time_cells;
    Split( data_cells[1], ':', time_cells, false );

    int hour      = ToInt(time_cells[0]);
    int minutes   = ToInt(time_cells[1]);
    int seconds   = ToInt(time_cells[2]);
    string am_pm  = time_cells[3];

    if( hour == 12 ) hour = hour-12;
    if( am_pm == "PM" ) hour = hour+12;

    int time_index = cInstant( ToString(hour) + string(":") + ToString(minutes) +
                               string(":") + ToString(seconds) ) /
                     k_time_slice_length;

    if( time_index < k_time_slice_offset )
        time_index = time_index + k_time_slices_size;

    if( stop_index == 0 && entry_exit == eEntry )
        time_index++;
    if( stop_index == 0 && entry_exit == eExit )
        time_index--;
    if( entry_exit == eExit )
        time_index--;

    time_index -= k_time_slice_offset;

    // set loading
    if( time_index > 0 && time_index < k_time_slices_size )
        loading[time_index][stop_index][entry_exit] += load;

    /*
    oss_log << EDayName(day) << tab;

    oss_log << stop_index << tab;

    oss_log << TimeSliceDuration(time_index) << tab;

    oss_log << EEntryExitName( entry_exit ) << tab;

    oss_log << load << ret;
    */
}
```

```
        // make loads
        float loads[k_time_slices_size][k_stops_size][k_stops_size];
        for_all_time_slices
        {
            for_all_stops_from
            {
                // total exit load for this from stop
                float all_to_exits_load = 0;
                for( int to_exit = 0; to_exit < k_stops_size; to_exit++ )
                    if( to_exit != from )
                        all_to_exits_load += loading[time_slice][to_exit][eExit];

                for_all_stops_to
                {
                    if( from != to && time_slice >= 6 && time_slice <= 85 )
                    {
                        // calculate load
                        loads[time_slice][from][to] = 0;
                        if( all_to_exits_load != 0 )
                        {
                            loads[time_slice][from][to]
                            = loading[time_slice][from][eEntry] *
                              (loading[time_slice][to][eExit]/all_to_exits_load);
                        }
                    }
                    else
                    {
                        loads[time_slice][from][to] = 0;
                    }
                }
            }
        }

        WriteLoads( loads, string("loads ") + EDayName(day) + string(".txt") );
    }

    WriteFile( "oss_log.txt", oss_log.str() );
}
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * duration.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * duration.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * duration.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#pragma once

#include "instant.h"

class cDuration
{
    public :

                            cDuration();
                            cDuration( cInstant iStart, cInstant iEnd );
                            cDuration( string iString );

        cInstant            Start() const;
        cInstant            End() const;

        void                AdjustStart( cInstant adjustment );
        void                AdjustEnd( cInstant adjustment );

        bool                IsInOpen( cInstant iInstant ) const;

        cInstant            Centre() const;
        cInstant            Width() const;

        bool                operator == ( const cDuration& duration ) const;

        friend              ostream& operator << ( ostream& os, const cDuration& d );

        static bool         Overlapping( cDuration a, cDuration b );
        static cDuration    Overlap( cDuration a, cDuration b );

    private :

        cInstant            mStart;
        cInstant            mEnd;
};

int          TimeSliceIndex( cDuration iTimeSlice );
cDuration    TimeSliceDuration( int iIndex );
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * duration.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * duration.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * duration.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "duration.h"

#include "line.h"

cDuration::cDuration()
{
}

cDuration::cDuration( cInstant iStart, cInstant iEnd )
{
    mStart = min(iStart,iEnd);
    mEnd   = max(iStart,iEnd);
}

cDuration::cDuration( string iString )
{
    vector<string> cells;
    Split( iString, '-', cells, false );

    mStart = cInstant( cells[0] );
    mEnd = cInstant( cells[1] );
}

cInstant
cDuration::Start() const
{
    return mStart;
}

cInstant
cDuration::End() const
{
    return mEnd;
}

void
cDuration::AdjustStart( cInstant adjustment )
{
    mStart = mStart + adjustment;
}

void
cDuration::AdjustEnd( cInstant adjustment )
{
    mEnd = mEnd + adjustment;
}

bool
cDuration::IsInOpen( cInstant iTime ) const
{
    return ( (mStart < iTime) && (mEnd > iTime) );
}

cInstant
cDuration::Centre() const
{
    return cInstant( average( mStart, mEnd ) );
}
```

```
cInstant
cDuration::Width() const
{
    return (mEnd - mStart).Absolute();
}

bool cDuration::operator == ( const cDuration& d ) const
{
    return ( Start() == d.Start() && End() == d.End() );
}

ostream& operator << ( ostream& os, const cDuration& d )
{
    return os << d.mStart << " - " << d.mEnd;
}

bool
cDuration::Overlapping( cDuration a, cDuration b )
{
    bool are_overlapping = false;

    if( b.mStart <= a.mStart && b.mEnd >= a.mStart ||
        b.mStart <= a.mEnd   && b.mEnd >= a.mEnd   ||
        b.mStart <= a.mStart && b.mEnd >= a.mEnd   ||
        a.mStart <= b.mStart && a.mEnd >= b.mEnd   )
    {
        are_overlapping = true;
    }

    return are_overlapping;
}

cDuration
cDuration::Overlap( cDuration a, cDuration b )
{
    cDuration overlap;

    overlap.mStart  = ( a.mStart < b.mStart ? b.mStart : a.mStart );
    overlap.mEnd    = ( a.mEnd   > b.mEnd   ? b.mEnd   : a.mEnd   );

    return overlap;
}

int TimeSliceIndex( cDuration iTimeSlice )
{
    return (iTimeSlice.Start()-k_open_time)/k_time_slice_length;
}

cDuration TimeSliceDuration( int iIndex )
{
    return cDuration( cInstant((iIndex+k_time_slice_offset)*k_time_slice_length),
                      cInstant((iIndex+k_time_slice_offset+1)*k_time_slice_length) );
}
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * instant.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * instant.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * instant.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#pragma once

//
//    class
//

#include <string>
#include <vector>
#include <sstream>
using namespace std;

#include "utility.h"

class cInstant
{
    public :

                cInstant();
                explicit cInstant( float iSeconds );
                explicit cInstant( string iString );

        int     Hours() const;
        int     Minutes() const;
        float   Seconds() const;
        int     SecondsFloor() const;

        float    AsHours() const;
        float    AsMinutes() const;
        float    AsSeconds() const;

        cInstant     Absolute() const;
        cInstant     RoundToSeconds() const;

        cInstant     operator -  () const;

        cInstant&    operator += ( const cInstant& a );

        bool    operator <  ( const cInstant& a ) const;
        bool    operator >  ( const cInstant& a ) const;
        bool    operator >= ( const cInstant& a ) const;
        bool    operator <= ( const cInstant& a ) const;
        bool    operator == ( const cInstant& a ) const;
        bool    operator != ( const cInstant& a ) const;

        friend     cInstant     operator - ( const cInstant&, const cInstant& );
        friend     cInstant     operator + ( const cInstant&, const cInstant& );
        friend     cInstant     operator * ( int,             const cInstant& );
        friend     cInstant     operator * ( float,           const cInstant& );
        friend     cInstant     operator / ( const cInstant&, int             );
        friend     float        operator / ( const cInstant&, const cInstant& );

        friend    ostream& operator << ( ostream& os, const cInstant& instant );

        static    string LeadZero( string iNumber );

    private :

        float    mSeconds;
};
```

```
// a typedef and function object that assist in
// melding cInstant with an int so that a magnitude
// can be associated with an instant.

typedef pair<cInstant,int> pair_cinstant_int_t;

struct pair_cinstant_int_t_FO
{
    bool operator() (const pair_cinstant_int_t& a, const pair_cinstant_int_t& b)
    {
        return a.first<b.first;
    }
};

// constants

const cInstant k_zero_seconds = cInstant("00:00:00");
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * instant.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * instant.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * instant.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "instant.h"

#include <cmath>
using namespace std;

cInstant::cInstant()
{
    mSeconds = 0;
}

cInstant::cInstant( float iSeconds )
{
    mSeconds = iSeconds;
}

cInstant::cInstant( string iString )
{
    vector<string> data_cells;
    Split( iString, ':', data_cells, false );

    if( data_cells.size() == 2 )
    {
        iString = "00:" + iString;
        data_cells.erase( data_cells.begin(), data_cells.end() );
        Split( iString, ':', data_cells, false );
    }

    mSeconds =  ToInt(data_cells[0]) * 3600 +
                ToInt(data_cells[1]) * 60 +
                ToFloat(data_cells[2]);
}

int
cInstant::Hours() const
{
    return floor( mSeconds/3600 );
}

int
cInstant::Minutes() const
{
    return floor( (mSeconds-(Hours()*3600))/60 );
}

float
cInstant::Seconds() const
{
    return mSeconds-float(Hours()*3600)-float(Minutes()*60);
}

int
cInstant::SecondsFloor() const
{
    return floor(Seconds());
}
```

```
float
cInstant::AsHours() const
{
    return mSeconds/3600.0;
}

float
cInstant::AsMinutes() const
{
    return mSeconds/60.0;
}

float
cInstant::AsSeconds() const
{
    return mSeconds;
}

cInstant
cInstant::Absolute() const
{
    cInstant temp = *this;
    temp.mSeconds = abs(mSeconds);
    return(temp);
}

cInstant
cInstant::RoundToSeconds() const
{
    cInstant temp = *this;
    temp.mSeconds = round(mSeconds);
    return(temp);
}

cInstant
cInstant::operator - () const
{
    return cInstant(-mSeconds);
}

cInstant&
cInstant::operator += ( const cInstant& a )
{
    return ( *this = *this + a );
}

bool
cInstant::operator < ( const cInstant& a ) const
{
    return mSeconds < a.mSeconds;
}

bool
cInstant::operator > ( const cInstant& a ) const
{
    return mSeconds > a.mSeconds;
}

bool
cInstant::operator >= ( const cInstant& a ) const
{
    return mSeconds >= a.mSeconds;
}

bool
cInstant::operator <= ( const cInstant& a ) const
{
    return mSeconds <= a.mSeconds;
}

bool
cInstant::operator == ( const cInstant& a ) const
{
    return mSeconds == a.mSeconds;
}
```

```
bool
cInstant::operator != ( const cInstant& a ) const
{
    return mSeconds != a.mSeconds;
}

cInstant operator - ( const cInstant& a, const cInstant& b )
{
    return cInstant( a.mSeconds - b.mSeconds );
}

cInstant operator + ( const cInstant& a, const cInstant& b )
{
    return cInstant( a.mSeconds + b.mSeconds );
}

cInstant operator * ( int i, const cInstant& a )
{
    return cInstant( float(i) * a.mSeconds );
}

cInstant operator * ( float f, const cInstant& a )
{
    return cInstant( f * a.mSeconds );
}

cInstant operator / ( const cInstant& a, int i )
{
    return cInstant( a.mSeconds / float(i) );
}

float operator / ( const cInstant& a, const cInstant& b )
{
    return a.mSeconds / b.mSeconds;
}

ostream& operator << ( ostream& os, const cInstant& time )
{
    return     os << time.LeadZero(ToString(time.Hours()))
                  << ":"

                  << time.LeadZero(ToString(time.Minutes()))
                  << ":"

                  << ( time.Seconds() < 10 ? "0" : "" )
                  << ToString(time.Seconds());
}

string
cInstant::LeadZero( string iNumber )
{
    return ( iNumber.length() >= 2 ? iNumber : "0" + iNumber );
}
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * line.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * line.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * line.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#pragma once

#include "stop.h"

#include "utility.h"

//
//    consts
//

const int       k_stops_size                = 18,
                k_combs_size                = 65536,
                k_all_stops_combination     = k_combs_size-1,
                k_intermediate_stops_size   = k_stops_size-2,
                k_innermost_stop            = 0,
                k_outermost_stop_far        = k_stops_size-1,
                k_outermost_stop_near       = 13,
                k_spans_size                = k_stops_size-1,
                k_links_size                = (k_stops_size*k_stops_size),
                k_occupancy_levels_size     = 4,
                k_channels_size             = 4;

const cInstant  k_time_to_line_speed        = cInstant(31.6),
                k_time_from_line_speed      = cInstant(13.8),
                k_interstop_traffic_time    = cInstant(37.0574917648);
                                            //cInstant(48.833985133) cInstant(73.250977699)

const float     k_line_speed                = 60/3.6,
                k_distance_to_line_speed    = 203.96,
                k_distance_from_line_speed  = 115.00,
                k_bus_capacity              = 160,
                k_mid_point                 = 5903.5,//13141
                k_line_load                 = 186000.0,
                k_minimum_number_of_runs    = 3.0, // per 15 minutes
                k_maximum_channels          = 4.0,
                k_pax_board_alight_per_sec  = 3.333333333; // each board alight 0.3 seconds


const cInstant  k_open_time                 = cInstant("03:00:00"),
                k_close_time                = cInstant("27:00:00"),
                k_time_slice_length         = cInstant("00:15:00"),

                k_bus_door_open_time        = cInstant("00:00:03.5"),
                k_bus_door_close_time       = cInstant("00:00:04.9"),

                k_run_slack                 = cInstant("00:01:30"); // pause at outermost
                                                                    // station each run pair

const int       k_time_slices_size          = 96, // (k_close_time-k_open_time)
                                                  // / k_time_slice_length;
                k_time_slice_offset         = 12;


const int       k_turn_around_stop[ENearFarSize]
                = {k_outermost_stop_near,k_outermost_stop_far};
```

```
//
//    class
//

class cLine
{
    public :

        float       Distance( int iStop );

        int         OutermostStop() const;
        int         InnermostStop() const;

        int         Stop( EDirection iDirection, EFirstLast iFirstLast ) const;

        bool        IsFinalStop( EDirection iDirection, int iStop );

        bool        IsEarlierStop( EDirection iDirection, int iStop, int iTestStop );
        bool        IsLatterStop( EDirection iDirection, int iStop, int iTestStop );

        bool        IsPastStop( EDirection iDirection, int iStop ) const;

        int         NextStop( EDirection iDirection, int iStop );

        float       LineLength();

        int         Stop( string iName );
        string      StopName( int iIndex );

        bool        IsStopExternalIndex( int iExternalIndex );
        int         ExternalIndexToStop( int iExternalIndex );

        static      cLine&    Singleton();

                    ~cLine();

    private :

                        cLine();

        vector<cStop>   mStops;

        static cLine    *sSingleton;
};

cLine&    Line();
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * line.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * line.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * line.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "line.h"

#include "utility.h"

//
// singleton mechanics
//

cLine&    Line()
{
    return cLine::Singleton();
}

cLine *cLine::sSingleton = NULL;

cLine&
cLine::Singleton()
{
    if ( sSingleton == NULL )
        sSingleton = new cLine();
    return *sSingleton;
}

//
// class functioncs
//

float
cLine::Distance( int iStop )
{
    return mStops[iStop].Distance();
}

int
cLine::OutermostStop() const
{
    return k_stops_size-1;
}

int
cLine::InnermostStop() const
{
    return 0;
}

int
cLine::Stop( EDirection iDirection, EFirstLast iFirstLast ) const
{
    if( (iDirection == eInbound  && iFirstLast == eFirst) ||
        (iDirection == eOutbound && iFirstLast == eLast ) )
        return k_stops_size-1;

    if( (iDirection == eOutbound && iFirstLast == eFirst) ||
        (iDirection == eInbound  && iFirstLast == eLast ) )
        return 0;

    throw string ( "cStopManager::Stop" );

    return 0;
}
```

```
bool
cLine::IsFinalStop( EDirection iDirection, int iStop )
{
    if( iDirection == eInbound )
        return iStop == 0;

    if( iDirection == eOutbound )
        return iStop == (k_stops_size-1);

    throw string ("unknown direction");

    return false;
}

bool
cLine::IsEarlierStop( EDirection iDirection, int iStop, int iTestStop )
{
    if( iDirection == eInbound )
        return iTestStop > iStop;

    if( iDirection == eOutbound )
        return iTestStop < iStop;

    throw string ("unknown direction");

    return false;
}

bool
cLine::IsLatterStop( EDirection iDirection, int iStop, int iTestStop )
{
    if( iDirection == eInbound )
        return iTestStop < iStop;

    if( iDirection == eOutbound )
        return iTestStop > iStop;

    throw string ("unknown direction");

    return false;
}


bool
cLine::IsPastStop( EDirection iDirection, int iStop ) const
{
    if( ( iDirection == eInbound && iStop > 8 ) || ( iDirection == eOutbound && iStop < 9 ) )
        return true;

    return false;
}

int
cLine::NextStop( EDirection iDirection, int iStop )
{
    if( iDirection == eInbound )
        return iStop-1;

    if( iDirection == eOutbound )
        return iStop+1;

    throw string ("unknown direction");

    return -1;
}

float
cLine::LineLength()
{
    return max(mStops.front().Distance(),
               mStops.back().Distance()) - min(mStops.front().Distance(),
               mStops.back().Distance());
}
```

```
int
cLine::Stop( string iName )
{
    for_all( vector<cStop>, mStops, stop )
        if( iName == stop->Name() )
            return stop->Index();

    throw string ("no such stop");

    return -1;
}

string
cLine::StopName( int iStop )
{
    return mStops[iStop].Name();
}

bool
cLine::IsStopExternalIndex( int iExternalIndex )
{
    for_all( vector<cStop>, mStops, stop )
        if( iExternalIndex == stop->ExternalIndex() )
            return true;

    return false;
}

int
cLine::ExternalIndexToStop( int iExternalIndex )
{
    for_all( vector<cStop>, mStops, stop )
        if( iExternalIndex == stop->ExternalIndex() )
            return stop->Index();

    throw string ("no such stop");

    return -1;
}

//
// constructor
//

cLine::cLine()
{
    // read stop data
    vector<string> file_lines = GetLines( ReadFile("stops.txt") );

    // make stops
    int stop_index = 0;
    const_for_all( vector<string>, file_lines, file_line )
    {
        vector<string> data_cells = GetCells( *file_line );

        mStops.push_back( cStop( stop_index++, data_cells[0],
                                 ToFloat(data_cells[1]), ToFloat(data_cells[2]) ) );
    }
}

//
// destructor
//

cLine::~cLine()
{
    sSingleton = NULL;
}
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * link.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * link.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * link.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#pragma once

#include "line.h"

class cLink
{
    public :

                            cLink();
                            cLink( int iLink );
                            cLink( int mFrom, int mTo );

        int                 From() const;
        int                 To() const;

        float               Distance() const;
        cInstant            Time() const;

        bool                IsDirection( EDirection iDirection ) const;
        bool                IsDirectionExtended( EDirectionExtended iDirection ) const;
        bool                IsBetween( int iStop ) const;

        EDirection          Direction() const;
        float               SpanCount() const;

        static float        StaticDistance(int iFrom, int iTo);

        static bool         StaticIsDirection( int iFrom, int iTo, EDirection iDirection );
        static bool         StaticIsBetween( int iFrom, int iTo, int iStop );

        static EDirection   StaticDirection(int iFrom, int iTo);

    private :

        int        mLink;
};

void           ZeroLinkArray( float links[k_links_size] );
int            Link( int iFrom, int iTo );
EDirection     Direction( int iFrom, int iTo );
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * link.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * link.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * link.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "link.h"

#include "line.h"

#include <iostream>
#include <cmath>
using namespace std;

//
// class
//

cLink::cLink()
{
}

cLink::cLink( int iLink )
{
    mLink = iLink;
}

cLink::cLink( int mFrom, int mTo )
{
    mLink = (mFrom*k_stops_size)+mTo;
}

int
cLink::From() const
{
    return floor(float(mLink)/float(k_stops_size));
}

int
cLink::To() const
{
    return mLink - (From()*k_stops_size);
}

float
cLink::Distance() const
{
    return abs( Line().Distance(To()) - Line().Distance(From()) );
}

cInstant
cLink::Time() const
{
    cInstant full_speed_time = cInstant( (Distance() -( k_distance_to_line_speed +
                                          k_distance_from_line_speed ))/k_line_speed );

    return full_speed_time + k_time_to_line_speed +
           k_time_from_line_speed + (SpanCount() * k_interstop_traffic_time);
}

bool
cLink::IsDirection( EDirection iDirection ) const
{
    return StaticIsDirection(From(),To(),iDirection);
}
```

```
bool
cLink::IsDirectionExtended( EDirectionExtended iDirection ) const
{
    if( iDirection == eEitherDirection )
        return true;

    if( iDirection == eInboundExtended && StaticIsDirection(From(),To(),eInbound) )
        return true;

    if( iDirection == eOutboundExtended && StaticIsDirection(From(),To(),eOutbound) )
        return true;

    return false;
}

bool
cLink::IsBetween( int iStop ) const
{
    return StaticIsBetween(From(),To(),iStop);
}

EDirection
cLink::Direction() const
{
    return StaticDirection(From(),To());
}

float
cLink::SpanCount() const
{
    return abs(float(From()-To()));
}

bool
cLink::StaticIsDirection( int iFrom, int iTo, EDirection iDirection )
{
    if( iDirection == eInbound )
        return iTo < iFrom;

    if( iDirection == eOutbound )
        return iTo > iFrom;

    throw string ("unknown direction");

    return false;
}

bool
cLink::StaticIsBetween( int iFrom, int iTo, int iStop )
{
    if( (iFrom > iStop && iTo < iStop) || (iFrom > iStop && iTo < iStop) )
        return true;

    return false;
}

EDirection
cLink::StaticDirection(int iFrom, int iTo)
{
    if( StaticIsDirection(iFrom,iTo,eInbound) )
        return eInbound;

    if( StaticIsDirection(iFrom,iTo,eOutbound) )
        return eOutbound;

    throw string ( "unknown direction" );

    return eInbound;
}
```

```
//
// functions
//

void
ZeroLinkArray( float links[k_links_size] )
{
    for_all_links
        links[link] = 0;
}

int
Link( int iFrom, int iTo )
{
    return (iFrom*k_stops_size)+iTo;
}

EDirection
Direction( int iFrom, int iTo )
{
    return cLink::StaticDirection( iFrom, iTo );
}
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * load.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * load.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * load.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#pragma once

#include "line.h"

void ReadLoads(  float oLoads[k_time_slices_size][k_stops_size][k_stops_size],
                 string iFileName );
void WriteLoads( float iLoads[k_time_slices_size][k_stops_size][k_stops_size],
                 string iFileName );

void MakeLoads();
void OffsetLoadsToLineCentre( float oLoads[k_time_slices_size][k_stops_size][k_stops_size] );

float SpanLoad(  float iLinkLoads[k_links_size], vector<int> iLinkSpans[k_links_size],
                 vector<int>& iLinks, int iSpan );
float PeakLoad(  float iLinkLoads[k_links_size], vector<int> iLinkSpans[k_links_size],
                 vector<int>& iLinks );
float TotalLoad( float iLinkLoads[k_links_size] );
float TotalLoad( float iLinkLoads[k_links_size], EDirection iDirection );
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * load.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * load.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * load.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "load.h"

#include "link.h"
#include "span.h"
#include "duration.h"

#include <iostream>


void ReadLoads( float oLoads[k_time_slices_size][k_stops_size][k_stops_size],
                    string iFileName )
{
    // get array of strings from file
    const int num_rows       = k_time_slices_size+2;
    const int num_columns    = k_stops_size*k_stops_size+1;

    string cells[num_rows][num_columns];
    vector<string> file_lines = GetLines( ReadFile(iFileName) );

    for( int row = 0; row < num_rows; row++)
    {
        vector<string> file_cells = GetCells( file_lines[row] );
        for( int column = 0; column < num_columns; column++)
            cells[row][column] = file_cells[column];
    }

    // zero all loads
    for_all_time_slices
        for_all_stops_from
            for_all_stops_to
                oLoads[time_slice][from][to] = 0;

    // covert array of strings to loads
    for( int row = 2; row < num_rows; row++)
    {
        for( int column = 1; column < num_columns; column++)
        {
            oLoads[row-2]
                  [Line().Stop(cells[0][column])]
                  [Line().Stop(cells[1][column])] = ToFloat(cells[row][column]);
        }
    }
}

void WriteLoads( float iLoads[k_time_slices_size][k_stops_size][k_stops_size],
                    string iFileName )
{
    const int num_rows       = k_time_slices_size+2;
    const int num_columns    = k_stops_size*k_stops_size+1;

    ostringstream cells[num_rows][num_columns];

    cells[0][0] << "time";
    cells[1][0] << "time";

    for_all_time_slices
        cells[2+time_slice][0] << TimeSliceDuration(time_slice);
```

```
    for_all_stops_from
        for_all_stops_to
        {
            int column = from*k_stops_size + to + 1;

            cells[0][column] << Line().StopName(from);
            cells[1][column] << Line().StopName(to);

            for_all_time_slices
                cells[2+time_slice][column] << iLoads[time_slice][from][to];
        }

    ostringstream oss_loads;

    for( int row = 0; row < num_rows; row++)
        for( int column = 0; column < num_columns; column++)
        {
            oss_loads << cells[row][column].str();
            oss_loads << ( column != num_columns-1 ? tab : ret );
        }


    WriteFile( iFileName, oss_loads.str() );
}

void
MakeLoads()
{
    // zero average load array
    float average_loads[k_time_slices_size][k_stops_size][k_stops_size];
    for_all_time_slices
        for_all_stops_from
            for_all_stops_to
                average_loads[time_slice][from][to] = 0;

    // process each day
    float all_days_max_total_load = 0;
    for_all_days
    {
        // get day loads
        float day_loads[k_time_slices_size][k_stops_size][k_stops_size];
        ReadLoads( day_loads, string("loads ") + EDayName(day) + string(".txt") );

        // get total load for day
        float day_max_load = 0;
        for_all_time_slices
            for_all_stops_from
                for_all_stops_to
                    day_max_load += day_loads[time_slice][from][to];

        // set maximum load for all days
        all_days_max_total_load = max( all_days_max_total_load, day_max_load );

        // add day to average loads
        for_all_time_slices
            for_all_stops_from
                for_all_stops_to
                {
                    average_loads[time_slice][from][to]
                    += day_loads[time_slice][from][to]/5.0;
                }

        cout << EDayName(day)           << space
             << day_max_load            << space
             << all_days_max_total_load << endl;
    }

    // get total for average load
    float average_load_total = 0;
        for_all_time_slices
            for_all_stops_from
                for_all_stops_to
                    average_load_total += average_loads[time_slice][from][to];
```

```
    // scale average load up to k_line_load
    for_all_time_slices
        for_all_stops_from
            for_all_stops_to
            {
                average_loads[time_slice][from][to]
                = average_loads[time_slice][from][to] * ( k_line_load/average_load_total );
            }

    WriteLoads( average_loads, "loads.txt" );

    cout << average_load_total << endl;
}

void
OffsetLoadsToLineCentre( float oLoads[k_time_slices_size][k_stops_size][k_stops_size] )
{
    const cInstant k_approximate_full_line_time = cInstant("00:30:00");

    // zero the adjusted loads container
    float adjusted_loads[k_time_slices_size][k_stops_size][k_stops_size];
    for_all_time_slices
        for_all_stops_from
            for_all_stops_to
                adjusted_loads[time_slice][from][to] = 0;

    for_all_time_slices
    {
        //cout << time_slice << endl;

        for_all_stops_from
            for_all_stops_to
                if( from != to )
                {
                    // ratio from mid-point
                    float distance_ratio
                    = ( from > 8 ?
                        (Line().Distance(from)-k_mid_point)/Line().LineLength() :
                        (k_mid_point-Line().Distance(from))/Line().LineLength() );

                    // convert offset into time
                    cInstant offset
                    = cInstant( distance_ratio * k_approximate_full_line_time );

                    // negative time if in past
                    if( Line().IsPastStop(cLink(from,to).Direction(), from) )
                        offset = -offset;

                    // make a duration and adjust by the offset
                    cDuration offset_duration = TimeSliceDuration( time_slice );
                    offset_duration.AdjustStart( offset );
                    offset_duration.AdjustEnd(   offset );

                    // get adjusted load
                    for( int cand_time_slices = 0;
                         cand_time_slices < k_time_slices_size;
                         cand_time_slices++ )
                    {
                        if( cDuration::Overlapping( offset_duration,
                            TimeSliceDuration( cand_time_slices ) ) )
                        {
                            float time_ratio
                            = cDuration::Overlap( offset_duration,
                              TimeSliceDuration( cand_time_slices ) )
                              .Width()/k_time_slice_length;

                            adjusted_loads[time_slice][from][to]
                            += oLoads[cand_time_slices][from][to]*time_ratio;
                        }
                    }
                }
    }

    // copy back
    for_all_time_slices
        for_all_stops_from
            for_all_stops_to
                oLoads[time_slice][from][to] = adjusted_loads[time_slice][from][to];
}
```

```
float SpanLoad( float iLinkLoads[k_links_size], vector<int> iLinkSpans[k_links_size],
vector<int>& iLinks, int iSpan )
{
    float span_loads[k_spans_size];
    ZeroSpanArray(span_loads);

    for_all( vector<int>, iLinks, link )
        for_all( vector<int>, iLinkSpans[*link], span )
            span_loads[*span] += iLinkLoads[*link];

    return span_loads[iSpan];
};

float PeakLoad( float iLinkLoads[k_links_size], vector<int> iLinkSpans[k_links_size],
vector<int>& iLinks )
{
    float span_loads[k_spans_size];
    ZeroSpanArray(span_loads);

    for_all( vector<int>, iLinks, link )
        for_all( vector<int>, iLinkSpans[*link], span )
            span_loads[*span] += iLinkLoads[*link];

    float peak_load = 0;
    for_all_spans
        peak_load = max( span_loads[span], peak_load );

    return peak_load;
};

float TotalLoad( float iLinkLoads[k_links_size] )
{
    float total_load = 0;
    for_all_links
        total_load += iLinkLoads[link];

    return total_load;
};

float TotalLoad( float iLinkLoads[k_links_size], EDirection iDirection )
{
    float total_load = 0;
    for_all_links
        if( cLink(link).IsDirection(iDirection) )
            total_load += iLinkLoads[link];

    return total_load;
};
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * pax_group.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * pax_group.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * pax_group.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#pragma once

#include "link.h"
#include "duration.h"

class cPaxGroup
{
    public :

                            cPaxGroup();
                            cPaxGroup( float    iGroupSize,
                                       cInstant iArrive,
                                       cLink    iLink,
                                       cInstant iFromTime,
                                       cInstant iToTime );

                            cPaxGroup( string iDataLine );

        float               GroupSize() const;

        cInstant            ArriveAtStop() const;

        cLink               Link() const;

        bool                TravelsBetweenThisStopAndNext( int iStop ) const;

        float               Distance() const;

        bool                TravelsBetween( int iFrom, int iTo ) const;

        cInstant            Time( ETripTimeType iTimeType ) const;
        cDuration           Duration( ETripTimeType iTimeType ) const;

        bool                operator <  ( const cPaxGroup& pax_group ) const;

        friend    ostream&  operator << ( ostream& os, const cPaxGroup& pax_group );

    private :

        float               mGroupSize;

        cInstant            mArrive;

        cLink               mLink;

        cInstant            mToTime;
        cInstant            mFromTime;
};

vector<cPaxGroup>   ReadPaxGroups();
void                WritePaxGroups( vector<cPaxGroup> iPaxGroup );
void                AnalysePaxs( vector<cPaxGroup> iPaxGroup, EDirectionExtended iDirection );
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * pax_group.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * pax_group.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * pax_group.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "pax_group.h"

#include "line.h"

#include "line.h"

#include <iostream>

//
// class
//

cPaxGroup::cPaxGroup()
{
}

cPaxGroup::cPaxGroup( float iGroupSize, cInstant iArrive, cLink iLink,
                      cInstant iFromTime, cInstant iToTime )
{
    mGroupSize   = iGroupSize;
    mArrive      = iArrive;
    mLink        = iLink;
    mFromTime    = iFromTime;
    mToTime      = iToTime;
}

cPaxGroup::cPaxGroup( string iDataLine )
{
    vector<string> data_cells = GetCells( iDataLine );

    mGroupSize    = ToFloat(data_cells[0]);
    mArrive       = cInstant(data_cells[1]);

    mLink         = cLink( Line().Stop(data_cells[2]), Line().Stop(data_cells[4]) );

    if( data_cells[3] != "not set" )
        mFromTime = cInstant(data_cells[3]);
    if( data_cells[5] != "not set" )
        mToTime = cInstant(data_cells[5]);
}


float
cPaxGroup::GroupSize() const
{
    return mGroupSize;
}

cInstant
cPaxGroup::ArriveAtStop() const
{
    return mArrive;
}

cLink
cPaxGroup::Link() const
{
    return mLink;
}
```

```
bool
cPaxGroup::TravelsBetweenThisStopAndNext( int iStop ) const
{
    if( mLink.IsDirection(eOutbound) && mLink.From() <= iStop &&
        mLink.To() >= Line().NextStop(eOutbound,iStop) )
    {
        return true;
    }

    if( mLink.IsDirection(eInbound ) && mLink.From() >= iStop &&
        mLink.To() <= Line().NextStop(eInbound ,iStop) )
    {
        return true;
    }

    return false;
}

float
cPaxGroup::Distance() const
{
    return Link().Distance();
}

bool
cPaxGroup::TravelsBetween( int iFrom, int iTo ) const
{
    if( mLink.IsDirection(eOutbound) && (mLink.From() <= iFrom) && (mLink.To() >= iTo ) )
        return true;

    if( mLink.IsDirection(eInbound) &&  (mLink.From() >= iFrom) && (mLink.To() <= iTo ) )
        return true;

    return false;
}

cInstant
cPaxGroup::Time( ETripTimeType iTimeType ) const
{
    if( iTimeType == eJourney )
        return mToTime - mArrive;
    if( iTimeType == eTransit )
        return mToTime - mFromTime;
    if( iTimeType == eWait )
        return mFromTime - mArrive;

    throw string ( "unknown trip time type" );

    return cInstant();
}

cDuration
cPaxGroup::Duration( ETripTimeType iTimeType ) const
{
    if( iTimeType == eJourney )
        return cDuration( mArrive, mToTime );
    if( iTimeType == eTransit )
        return cDuration( mFromTime, mToTime );
    if( iTimeType == eWait )
        return cDuration( mArrive, mFromTime );

    throw string ( "unknown trip time type" );

    return cDuration();
}

bool cPaxGroup::operator < ( const cPaxGroup& pax_group ) const
{
    return mArrive < pax_group.mArrive;
}
```

```
ostream& operator << ( ostream& os, const cPaxGroup& pax_group )
{
    ostringstream oss;

    oss << pax_group.mGroupSize << tab;
    oss << pax_group.mArrive    << tab;

    // from
    oss << Line().StopName(pax_group.mLink.From()) << tab;
    if( true )
        oss << pax_group.mFromTime;
    else
        oss << "not set";
    oss << tab;

    // to
    oss << Line().StopName(pax_group.mLink.To()) << tab;
    if( true )
        oss << pax_group.mToTime;
    else
        oss << "not set";
    oss << tab;

    return os << oss.str();
}

//
// functions
//

vector<cPaxGroup> ReadPaxGroups()
{
    vector<string> file_lines = GetLines( ReadFile("pax_groups.txt") );

    vector<cPaxGroup> pax_groups;
    const_for_all( vector<string>, file_lines, file_line )
        pax_groups.push_back( *file_line );

    return pax_groups;
}

void WritePaxGroups( vector<cPaxGroup> iPaxGroups )
{
    ostringstream oss;

    const_for_all( vector<cPaxGroup>, iPaxGroups, pax_group )
        oss << *pax_group << ret;

    WriteFile( "pax_groups.txt", oss.str() );
}

void AnalysePaxs( vector<cPaxGroup> iPaxGroups, EDirectionExtended iDirection )
{
    cout << "make " << EDirectionExtendedName(iDirection) << " pax analysis" << endl;

    ostringstream oss_pax_time[ETripTimeTypeSize];
    ostringstream oss_pax_speed[ETripTimeTypeSize];
    ostringstream oss_pax_distance[ETripTimeTypeSize];

    // for each trip time types
    for( int ttt = 0; ttt < ETripTimeTypeSize ; ttt++ )
    {
        oss_pax_time[ETripTimeType(ttt)]     << EDirectionExtendedName(iDirection) << ret;
        oss_pax_speed[ETripTimeType(ttt)]    << EDirectionExtendedName(iDirection) << ret;
        oss_pax_distance[ETripTimeType(ttt)] << EDirectionExtendedName(iDirection) << ret;
    }

    for_all_time_slices
    {
        cout << TimeSliceDuration(time_slice) << endl;

        for( int ttt = 0; ttt < ETripTimeTypeSize ; ttt++ )
        {
            float       time_slice_distance    = 0;
            cInstant    time_slice_time        = cInstant(0);
            float       time_slice_pax         = 0;
```

```
        // for each pax group
        const_for_all( vector<cPaxGroup>, iPaxGroups, pax_group )
        {
            // correct direction
            if( pax_group->Link().IsDirectionExtended(iDirection) )
            {
                // if pax group is in the time slice
                if( cDuration::Overlapping( TimeSliceDuration(time_slice),
                                            pax_group->Duration(ETripTimeType(ttt)) ) )
                {
                    float overlap_ratio
                    = cDuration::Overlap(TimeSliceDuration(time_slice)
                      pax_group->Duration(ETripTimeType(ttt)))
                      .Width().AsSeconds() /
                      pax_group->Duration(ETripTimeType(ttt)).Width().AsSeconds();

                    time_slice_distance += pax_group->GroupSize() *
                                           pax_group->Distance() * overlap_ratio;

                    time_slice_time
                    += pax_group->GroupSize() *
                       cDuration::Overlap(TimeSliceDuration(time_slice),
                                          pax_group->Duration(ETripTimeType(ttt))).Width();

                    time_slice_pax += overlap_ratio * pax_group->GroupSize();
                }
            }
        }

        // add distance data for this time slice
        if( time_slice_distance != 0 )
            oss_pax_distance[ETripTimeType(ttt)] << time_slice_distance/1000.0 << ret;
        else
            oss_pax_distance[ETripTimeType(ttt)] << ret;

        // add time data for this time slice
        if( time_slice_time != k_zero_seconds )
        {
            oss_pax_time[ETripTimeType(ttt)]
            << cInstant(time_slice_time).AsMinutes()/time_slice_pax << ret;
        }
        else
            oss_pax_time[ETripTimeType(ttt)] << ret;

        // add speed data for this time slice
        if( time_slice_time != k_zero_seconds )
        {
            oss_pax_speed[ETripTimeType(ttt)]
            << (time_slice_distance/1000.0)/(time_slice_time.AsSeconds()/3600.0) << ret;
        }
        else
            oss_pax_speed[ETripTimeType(ttt)] << ret;
    }
}

// write out time slice averages
for( int ttt = 0; ttt < ETripTimeTypeSize ; ttt++ )
{
    string file_name_end =  ETripTimeTypeName(ETripTimeType(ttt)) + space +
                            EDirectionExtendedName(iDirection) + ".txt";

    if( ETripTimeType(ttt) == eTransit )
        WriteFile( "paxs distance " + file_name_end,
                   oss_pax_distance[ETripTimeType(ttt)].str() );

    WriteFile( "paxs time " + file_name_end, oss_pax_time[ETripTimeType(ttt)].str() );

    if( ETripTimeType(ttt) != eWait )
        WriteFile( "paxs speed " + file_name_end,
                   oss_pax_speed[ETripTimeType(ttt)].str() );
}

// make overall analysis
float       overall_distance        = 0; // metres
cInstant    overall_journey_time    = cInstant(0),
            overall_transit_time    = cInstant(0),
            overall_wait_time       = cInstant(0);
float       overall_pax_size        = 0;
```

```
    // for each pax group
    const_for_all( vector<cPaxGroup>, iPaxGroups, pax_group )
    {
        // correct direction
        if( pax_group->Link().IsDirectionExtended(iDirection) )
        {
            overall_distance        +=    pax_group->GroupSize() * pax_group->Distance();
            overall_journey_time    +=    pax_group->GroupSize() * pax_group->Time(eJourney);
            overall_transit_time    +=    pax_group->GroupSize() * pax_group->Time(eTransit);
            overall_wait_time       +=    pax_group->GroupSize() * pax_group->Time(eWait);
            overall_pax_size        +=    pax_group->GroupSize();
        }
    }

    // output overall analysis
    ostringstream oss_pax_totals;

    oss_pax_totals    << "average wait time" << tab
                      << (overall_wait_time/overall_pax_size).RoundToSeconds() << ret;

    oss_pax_totals    << "average transit time" << tab
                      << (overall_transit_time/overall_pax_size).RoundToSeconds() << ret;

    oss_pax_totals    << "average journey time" << tab
                      << (overall_journey_time/overall_pax_size).RoundToSeconds() << ret;

    oss_pax_totals    << "average transit speed" << tab
                      << (overall_distance/1000.0)/(overall_transit_time.AsSeconds()/3600.0)
                      << ret;

    oss_pax_totals    << "average journey speed" << tab
                      << (overall_distance/1000.0)/(overall_journey_time.AsSeconds()/3600.0)
                      << ret;

    oss_pax_totals    << "total distance" << tab << overall_distance/1000.0 << ret;

    WriteFile( "paxs overall " +
               EDirectionExtendedName(iDirection) + ".txt", oss_pax_totals.str() );
}
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * run.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * run.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * run.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#pragma once

#include "stop.h"
#include "line.h"
#include "pax_group.h"

class cRun
{
    public :

                            cRun();

                            cRun(    EDirection iDirection, cInstant iCentreTime,
                                     int iInnermostStop, int iOutermostStop );
                            cRun(    EDirection iDirection, cInstant iFromTime,
                                     cInstant iToTime, vector<int> iStops );
                            cRun(    EDirection iDirection, cInstant iCentreTime,
                                     cInstant iPaxTimeWidth, vector<int> iStops,
                                     float iLoads[k_links_size] );

        int                 Stop( EInnerOuter iInnerOuter ) const;
        int                 Stop( EFromTo     iFromTo     ) const;

        bool                StopsAt( int iStop ) const;

        EDirection          Direction() const;
        bool                IsDirection( EDirection iDirection ) const;
        bool                IsDirectionExtended( EDirectionExtended iDirection ) const;

        cInstant            Time( EFromTo iFromTo ) const;

        vector<int>         Stops() const;
        void                AddStop( int iStop );
        void                EraseAllStops();

        bool                IsPaired() const;
        void                MarkAsPaired();
        void                MarkAsUnpaired();

        float               Load( cLink iLink ) const;

        float               PaxCount( int iStop, EFromTo iFromTo ) const;
        float               PaxCount() const;

        float               PaxTotal() const;
        cInstant            Time( int iStop, bool iAddSlackTime = false ) const;

        void                MakePaxs( vector<cPaxGroup>& iPaxGroups );

        cDuration           Duration() const;
        float               Distance() const;

        float               PaxDistance() const;

        float               Occupancy() const;

        bool                operator <  ( const cRun& v ) const;

        friend    ostream&  operator << ( ostream& os, const cRun& run );
```

```
    private :

        vector<int>         mStops;

        EDirection          mDirection;
        cInstant            mCentreTime;
        cInstant            mPaxTimeWidth;

        bool                mFromToTimeSet;
        cInstant            mFromTime;
        cInstant            mToTime;

        bool                mPaired;

        float               mLoads[k_links_size];

};

vector<cRun>    ReadRuns();
void            WriteRuns( vector<cRun> iRuns );
void            AnalyseRuns( vector<cRun> iRuns, EDirectionExtended iDirection );
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * run.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * run.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * run.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "run.h"

#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

//
// class
//

cRun::cRun()
{
}

cRun::cRun( EDirection iDirection, cInstant iCentreTime,
            int iInnermostStop, int iOutermostStop )
{
    mDirection = iDirection;
    mCentreTime = iCentreTime;

    mFromToTimeSet = false;

    mPaired = false;

    mStops.push_back( iInnermostStop );
    mStops.push_back( iOutermostStop );

    for_all_links
        mLoads[link] = 0;
}

cRun::cRun( EDirection iDirection, cInstant iFromTime, cInstant iToTime, vector<int> iStops )
{
    mDirection = iDirection;

    mFromToTimeSet = true;
    mFromTime = iFromTime;
    mToTime = iToTime;

    mPaired = false;

    copy_all( iStops, mStops );

    for_all_links
        mLoads[link] = 0;
}
```

```
cRun::cRun( EDirection iDirection, cInstant iCentreTime, cInstant iPaxTimeWidth,
            vector<int> iStops, float iLoads[k_links_size] )
{
    mDirection    = iDirection;
    mCentreTime   = iCentreTime;
    mPaxTimeWidth = iPaxTimeWidth;

    mFromToTimeSet = false;

    mPaired = false;

    copy_all( iStops, mStops );

    for_all_links
        mLoads[link] = iLoads[link];
}

int
cRun::Stop( EInnerOuter iInnerOuter ) const
{
    return ( iInnerOuter == eInner ? *min_element(all(mStops)) : *max_element(all(mStops)) );
}

int
cRun::Stop( EFromTo iFromTo ) const
{
    if( IsDirection(eInbound) )
        return ( iFromTo == eFrom ? Stop(eOuter) : Stop(eInner) );
    else
        return ( iFromTo == eFrom ? Stop(eInner) : Stop(eOuter) );
}

bool
cRun::StopsAt( int iTimeStop ) const
{
    bool stops_at = false;
    const_for_all( vector<int>, mStops, stop )
        if( iTimeStop == *stop )
            stops_at = true;

    return stops_at;
}

EDirection
cRun::Direction() const
{
    return mDirection;
}

bool
cRun::IsDirection( EDirection iDirection ) const
{
    return mDirection == iDirection;
}

bool
cRun::IsDirectionExtended( EDirectionExtended iDirection ) const
{
    if( iDirection == eEitherDirection )
        return true;

    if( iDirection == eInboundExtended && mDirection == eInbound )
        return true;

    if( iDirection == eOutboundExtended && mDirection == eOutbound )
        return true;

    return false;
}

cInstant
cRun::Time( EFromTo iFromTo ) const
{
    return Time(Stop(iFromTo));
}
```

```
vector<int>
cRun::Stops() const
{
    return mStops;
}

void
cRun::AddStop( int iStop )
{
    mStops.push_back( iStop );
}

void
cRun::EraseAllStops()
{
    erase_all( mStops );
}

bool
cRun::IsPaired() const
{
    return mPaired;
}

void
cRun::MarkAsPaired()
{
    mPaired = true;
}

void
cRun::MarkAsUnpaired()
{
    mPaired = false;
}

float
cRun::Load( cLink iLink ) const
{
    return mLoads[(iLink.From()*k_stops_size)+iLink.To()];
}

float
cRun::PaxCount( int iStop, EFromTo iFromTo ) const
{
    float pax_count = 0;

    for_all_links
    {
        if( iFromTo == eFrom && cLink(link).From() == iStop )
            pax_count += mLoads[link];
        if( iFromTo == eTo   && cLink(link).To()   == iStop )
            pax_count += mLoads[link];
    }

    return pax_count;
}

float
cRun::PaxCount() const
{
    float total_load = 0;

    for_all_links
        total_load += mLoads[link];

    return total_load;
}

float
cRun::PaxTotal() const
{
    float pax_count = 0;

    for_all_links
        pax_count += mLoads[link];

    return pax_count;
}
```

```
cInstant
cRun::Time( int iStop, bool iAddSlackTime ) const
{
    if( !StopsAt( iStop ) )
        return cInstant(0);

    if( mFromToTimeSet )
    {
        if( !( Stop(eFrom) == iStop || Stop(eTo) == iStop ) )
            throw string ("stop unknown");

        return ( Stop(eFrom) == iStop ? mFromTime : mToTime );
    }

    cInstant time = mCentreTime;

    vector<int> stops_from_midpoint;
    const_for_all( vector<int>, mStops, stop )
    {
        if( iStop < 9 && ( *stop < 9 && *stop >= iStop ) )
            stops_from_midpoint.push_back( *stop );

        if( iStop > 8 && ( *stop > 8 && *stop <= iStop ) )
            stops_from_midpoint.push_back( *stop );
    }

    long first_stop_from_mid_point = ( iStop < 9 ? 8 : 9 );

    sort_all( stops_from_midpoint );
    if( iStop < 9 )
        reverse_all( stops_from_midpoint );

    // past or future
    bool is_past = Line().IsPastStop( mDirection, iStop );

    cInstant time_offset = cInstant(0);
    for_all( vector<int>, stops_from_midpoint, stop )
    {
        if( stops_from_midpoint.begin() == stop )
        {
            time_offset
            += cInstant((abs(k_mid_point - Line().Distance(*stop)) -
               (is_past?k_distance_to_line_speed:k_distance_from_line_speed))/k_line_speed) +
               (is_past?k_time_to_line_speed:k_time_from_line_speed) +
               (float((abs(*stop - first_stop_from_mid_point) + 0.5)) *
                k_interstop_traffic_time );

            time_offset += (is_past?k_bus_door_close_time:k_bus_door_open_time);

            time_offset
            += cInstant(PaxCount(*stop,is_past?eFrom:eTo)/k_pax_board_alight_per_sec);
        }

        vector<int>::iterator next = stop; next++;
        if( next != stops_from_midpoint.end() )
        {
            time_offset
            += cInstant(PaxCount(*stop,is_past?eTo:eFrom)/k_pax_board_alight_per_sec);

            time_offset += (is_past?k_bus_door_open_time:k_bus_door_close_time);
            time_offset += cLink( *stop, *next ).Time();
            time_offset += (is_past?k_bus_door_close_time:k_bus_door_open_time);

            time_offset
            += cInstant(PaxCount(*next,is_past?eFrom:eTo)/k_pax_board_alight_per_sec);
        }
    }

    cInstant the_stop_time = mCentreTime + cInstant( is_past?-time_offset:time_offset );

    // add slack time
    if( iAddSlackTime && iStop == Stop(eOuter) && IsDirection(eOutbound) )
        the_stop_time += k_run_slack;

    return the_stop_time;
}
```

```
void
cRun::MakePaxs( vector<cPaxGroup>& iPaxGroups )
{
    for_all_links
        if( mLoads[link] != 0 )
            iPaxGroups.push_back( cPaxGroup( mLoads[link],
                                  Time(cLink(link).From()) - (mPaxTimeWidth/2.0), cLink(link),
                                  Time(cLink(link).From()), Time(cLink(link).To()) ) );
}

cDuration
cRun::Duration() const
{
    return cDuration( Time(Stop(eInner)), Time(Stop(eOuter)) );
}

float
cRun::Distance() const
{
    return cLink(Stop(eInner),Stop(eOuter)).Distance();
}

float
cRun::PaxDistance() const
{
    float pax_distance = 0;

    for_all_stops_from
        for_all_stops_to
            if( from != to )
                pax_distance += mLoads[Link(from,to)] * cLink(from,to).Distance();

    return pax_distance;
}

float
cRun::Occupancy() const
{
    return PaxDistance() / (Distance()*k_bus_capacity);
}

bool
cRun::operator < ( const cRun& v ) const
{
    return Time(k_innermost_stop) < v.Time(k_innermost_stop);
}

ostream& operator << ( ostream& os, const cRun& run )
{
    ostringstream oss;

    oss << run.mDirection      << endl;
    oss << run.mCentreTime     << endl;
    oss << run.mPaxTimeWidth   << endl;

    oss << run.mFromToTimeSet  << endl;
    oss << run.mFromTime       << endl;
    oss << run.mToTime         << endl;

    // Stops
    vector<int> the_stops = run.Stops();
    for_all( vector<int>, the_stops, the_stop )
    {
        vector<int>::iterator next = the_stop; next++;
        oss << *the_stop;
        if( next != the_stops.end() )
            oss << tab;
    }
    oss << endl;
```

```
    // Loads
    for_all_stops_from
    {
        for_all_stops_to
        {
            oss << run.Load(cLink(from,to));
            if( to+1 != k_stops_size )
                oss << tab;
        }
        oss << endl;
    }

    return os << oss.str();
}

//
// functions
//

vector<cRun> ReadRuns()
{
    vector<cRun> runs;

    vector<string> file_lines = GetLines( ReadFile("run_groups.txt") );
    vector< vector<string> > file_cells;
    for_all( vector<string>, file_lines, file_line )
        file_cells.push_back( GetCells(*file_line) );

    int num_runs = file_lines.size()/(k_stops_size+7);

    for( int run = 0; run < num_runs; run++ )
    {
        EDirection  dir                 = EDirection(ToInt(file_cells[(run*25)+0][0]));
        cInstant    centre_time         = cInstant(file_cells[(run*25)+1][0]);
        cInstant    pax_time_width      = cInstant(file_cells[(run*25)+2][0]);

        bool        from_to_time_set    = ToInt(file_cells[(run*25)+3][0]);
        cInstant    from_time           = cInstant(file_cells[(run*25)+4][0]);
        cInstant    to_time             = cInstant(file_cells[(run*25)+5][0]);

        vector<int> stops;
        for( int stop = 0; stop < file_cells[(run*25)+6].size(); stop++ )
            stops.push_back( ToInt(file_cells[(run*25)+6][stop]) );

        float loads[k_links_size];
        for_all_stops_from
            for_all_stops_to
                loads[(from*k_stops_size)+to] = ToFloat( file_cells[(run*25)+7+from][to] );

        if( from_to_time_set )
            runs.push_back( cRun( dir, from_time, to_time, stops ) );
        else
            runs.push_back( cRun( dir, centre_time, pax_time_width, stops, loads ) );
    }

    return runs;
}

void WriteRuns( vector<cRun> iRuns )
{
    ostringstream oss;

    const_for_all( vector<cRun>, iRuns, run )
        oss << *run;

    WriteFile( "run_groups.txt", oss.str() );
}

void AnalyseRuns( vector<cRun> iRuns, EDirectionExtended iDirection )
{
    cout << "make " << EDirectionExtendedName(iDirection) << " run analysis" << endl;

    ostringstream oss_distance;
    ostringstream oss_duration;
    ostringstream oss_occupancy;
```

```
oss_distance  << EDirectionExtendedName(iDirection) << ret;
oss_duration  << EDirectionExtendedName(iDirection) << ret;
oss_occupancy << EDirectionExtendedName(iDirection) << ret;

for_all_time_slices
{
    cout << TimeSliceDuration(time_slice) << endl;

    float       time_slice_distance_run         = 0,
                time_slice_distance_places      = 0,
                time_slice_distance_paxs        = 0;
    cInstant    time_slice_time                 = cInstant(0);

    // for each run
    const_for_all( vector<cRun>, iRuns, run )
    {
        // correct direction
        if( run->IsDirectionExtended(iDirection) )
        {
            // if run is in the time slice
            if( cDuration::Overlapping( TimeSliceDuration(time_slice), run->Duration() ) )
            {
                float overlap_ratio
                = cDuration::Overlap(TimeSliceDuration(time_slice),
                                     run->Duration()).Width().AsSeconds() /
                  run->Duration().Width().AsSeconds();

                time_slice_distance_run += run->Distance() * overlap_ratio;

                time_slice_distance_places
                += run->Distance() * overlap_ratio * k_bus_capacity;

                time_slice_distance_paxs += run->PaxDistance() * overlap_ratio;

                time_slice_time
                += cDuration::Overlap(TimeSliceDuration(time_slice),run->Duration())
                   .Width();
            }
        }
    }

    // add distance data for this time slice
    if( time_slice_distance_run != 0 )
        oss_distance << time_slice_distance_run/1000.0 << ret;
    else
        oss_distance << ret;

    // add time data for this time slice
    if( time_slice_time != cInstant(0) )
        oss_duration << time_slice_time.AsSeconds() << ret;
    else
        oss_duration << ret;

    // add occupancy data for this time slice (as percent)
    if( time_slice_distance_places != 0 )
    {
        oss_occupancy
        << time_slice_distance_paxs/time_slice_distance_places * 100.0 << ret;
    }
    else
        oss_occupancy << ret;
}

WriteFile( "runs distance "
           + EDirectionExtendedName(iDirection) + ".txt", oss_distance.str()  );

WriteFile( "runs time "
           + EDirectionExtendedName(iDirection) + ".txt", oss_duration.str()  );

WriteFile( "runs occupancy "
           + EDirectionExtendedName(iDirection) + ".txt", oss_occupancy.str() );
```

```
    // make overall analysis
    float       overall_distance         = 0,
                overall_distance_paxs    = 0,
                overall_distance_places  = 0;
    cInstant    overall_duration         = cInstant(0);

    // for each run
    const_for_all( vector<cRun>, iRuns, run )
    {
        // correct direction
        if( run->IsDirectionExtended(iDirection) )
        {
            overall_distance            += run->Distance();
            overall_duration            += run->Duration().Width();
            overall_distance_paxs       += run->PaxDistance();
            overall_distance_places     += run->Distance() * k_bus_capacity;
        }
    }

    // output overall analysis
    ostringstream oss_overall;

    oss_overall << "total distance" << tab
                << overall_distance/1000.0 << ret;

    oss_overall << "total duration" << tab
                << overall_duration << ret;

    oss_overall << "total pax distance" << tab
                << overall_distance_paxs/1000.0 << ret;

    oss_overall << "total places distance" << tab
                << overall_distance_places/1000.0 << ret;

    oss_overall << "occupancy" << tab
                << overall_distance_paxs/overall_distance_places << ret;

    WriteFile( "runs overall " + EDirectionExtendedName(iDirection)
               + ".txt", oss_overall.str() );
}
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * span.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * span.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * span.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#pragma once

#include "line.h"

class cSpan
{
    public :

        cSpan();
        cSpan( int iIndex );

    private :

        int                mIndex;
};

void ZeroSpanArray( float spans[k_spans_size] );
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * span.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * span.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * span.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "span.h"

//
// class
//

cSpan::cSpan()
{
}

cSpan::cSpan( int iIndex )
{
    mIndex = iIndex;
}

//
// functions
//

void
ZeroSpanArray( float spans[k_spans_size] )
{
    for_all_spans
        spans[span] = 0;
}
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * stop.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * stop.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * stop.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#pragma once

#include <string>
#include <vector>
#include <sstream>
using namespace std;

#include "instant.h"

class cStop
{
    public :

        cStop( int iIndex, string iName, float iDistance, float iExternalIndex );

        int      Index() const;
        string   Name() const;
        float    Distance() const;
        float    ExternalIndex() const;

    private :

        int      mIndex;
        string   mName;
        float    mDistance;
        float    mExternalIndex;
        int      mStartRuns;
};

string PrettyPrintStops( vector<int> iStops );
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * stop.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * stop.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * stop.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "stop.h"

#include "line.h"

#include <algorithm>
using namespace std;

//
// class
//

cStop::cStop( int iIndex, string iName, float iDistance, float iExternalIndex )
{
    mIndex           = iIndex;
    mName            = iName;
    mDistance        = iDistance;
    mExternalIndex   = iExternalIndex;
}

int
cStop::Index() const
{
    return mIndex;
}

string
cStop::Name() const
{
    return mName;
}

float
cStop::Distance() const
{
    return mDistance;
}

float
cStop::ExternalIndex() const
{
    return mExternalIndex;
}
```

```
//
// functions
//

string
PrettyPrintStops( vector<int> iStops )
{
    ostringstream oss_stops;

    sort_all(iStops);
    reverse_all(iStops);

    for( int every_stop = k_outermost_stop_far; every_stop >= k_innermost_stop; every_stop-- )
    {
        bool is_stopping = false;
        for_all( vector<int>, iStops, stop )
            if ( every_stop == *stop )
                is_stopping = true;

        if( is_stopping )
            oss_stops << every_stop << space;
        else
        {
            oss_stops << "-";
            if( every_stop >= 10 )
                oss_stops << "-";
            oss_stops << space;
        }
    }

    return oss_stops.str();
}
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * utility.h
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * utility.h is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * utility.h is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#pragma once

#include <string>
#include <vector>
#include <sstream>
using namespace std;

// basic characters

#define tab   '\t'
#define ret   '\n'
#define space ' '

// data getters

void Split( string iStr, char iSep, vector<string> &oList, bool discard_empties );

vector<string> GetLines( string iLines );
vector<string> GetCells( string iLine );

string RecombineLines( vector<string> iLines );
string RecombineCells( vector<string> iCells );

// data converters

int      ToInt( string iString );
float    ToFloat( string iString );

template <class T>
string ToString( const T &t )
{
    ostringstream oss;
    oss << t;
    return oss.str();
}

// STL

#define for_all( TYPE, COL, ITER ) \
    for ( TYPE::iterator ITER = COL.begin(); !(ITER == COL.end()); ITER++ )

#define const_for_all( TYPE, COL, ITER ) \
    for ( TYPE::const_iterator ITER = COL.begin(); !(ITER == COL.end()); ITER++ )

#define sort_all( COL ) sort ( COL.begin(), COL.end() )
#define reverse_all( COL ) reverse ( COL.begin(), COL.end() )
#define copy_all( COL_1, COL_2 ) copy ( COL_1.begin(), COL_1.end(), back_inserter(COL_2) )
#define contains( COL_1, ELEMENT ) find ( COL_1.begin(), COL_1.end(), ELEMENT ) != COL_1.end()
#define erase_all( COL_1 ) COL_1.erase ( COL_1.begin(), COL_1.end() )

#define next_combination_all( COL_1, COL_2 ) \
       next_combination ( COL_1.begin(), COL_1.end(), COL_2.begin(), COL_2.end() )

#define all( COL ) COL.begin(), COL.end()
```

```
// loops - forward

#define for_all_stops           for( int stop = 0; stop < k_stops_size; stop++ )
#define for_all_spans           for( int span = 0; span < k_spans_size; span++ )
#define for_all_links           for( int link = 0; link < k_links_size; link++ )
#define for_all_days            for( int day = 0;  day  < 5;            day++  )

#define for_all_channels        for( int channel = 0;
                                     channel < k_channels_size;
                                     channel++  )

#define for_all_combinations    for( int combination = 0;
                                     combination < k_combs_size;
                                     combination++  )

#define for_all_stops_from      for( int from = 0;      from < k_stops_size; from++ )
#define for_all_stops_to        for( int to = 0;        to   < k_stops_size; to++   )
#define for_both_directions     for( int direction = 0; direction < 2;       direction++ )
#define for_both_near_and_far   for( int near_far  = 0; near_far  < 2;       near_far++  )

#define for_all_time_slices     for( int time_slice = 0;
                                     time_slice < k_time_slices_size;
                                     time_slice++ )

// loops - reverse

#define for_all_express_channels_reverse    for( int channel = k_channels_size-1;
                                                 channel > 0; channel--  )

// file access

string  ReadFile( const string iFileName );
void    WriteFile( const string iFileName, const string iData );

// general

bool IsOdd( int iValue );
bool IsEven( int iValue );
bool IsWhole( float iValue );

float RandomLevel();
int   RandomChoice( int iSize );

string    BoolToString( bool b );
bool      StringToBool( string s );

// average

template < class C >
C average(C iA, C iB)
{
    return (iA+iB)/2;
}

// EDirection Enum

enum EDirection
{
    eInbound     = 0,
    eOutbound    = 1
};

const int EDirectionSize = 2;

string EDirectionName( int iIndex );

// EDirectionExtended Enum

enum EDirectionExtended
{
    eInboundExtended    = 0,
    eOutboundExtended   = 1,
    eEitherDirection    = 2
};

const int EDirectionExtendedSize = 3;

string EDirectionExtendedName( int iIndex );
```

```
// EFirstLast Enum

enum EFirstLast
{
    eFirst   = 0,
    eLast    = 1
};

const int EFirstLastSize = 2;

string EFirstLastName( int iIndex );

// EArriveDepart Enum

enum EArriveDepart
{
    eArrive    = 0,
    eDepart    = 1
};

const int EArriveDepartSize = 2;

string EArriveDepartName( int iIndex );

// ETemporalType Enum

enum ETemporalType
{
    eBefore    = 0,
    eAfter     = 1
};

const int ETemporalTypeSize = 2;

string ETemporalTypeName( int iIndex );

// EMinMax Enum

enum EMinMax
{
    eMin    = 0,
    eMax    = 1
};

const int EMinMaxSize = 2;

string EMinMaxName( int iIndex );

// EFromToType Enum

enum EFromTo
{
    eFrom    = 0,
    eTo      = 1
};

const int EFromToSize = 2;

string EFromToName( int iIndex );

// EEntryExit Enum

enum EEntryExit
{
    eEntry   = 0,
    eExit    = 1
};

const int EEntryExitSize = 2;

string EEntryExitName( int iIndex );
```

```
// EBoardAlight Enum

enum EBoardAlight
{
    eBoard    = 0,
    eAlight   = 1
};

const int EBoardAlightSize = 2;

string EBoardAlightName( int iIndex );

// EPosNeg Enum

enum EPosNeg
{
    ePos = 0,
    eNeg = 1
};

const int EPosNegSize = 2;

string EPosNegName( int iIndex );

// EInnerOuter Enum

enum EInnerOuter
{
    eInner = 0,
    eOuter = 1
};

const int EInnerOuterSize = 2;

string EInnerOuterName( int iIndex );

// EDay Enum

enum EDay
{
    eMonday     = 0,
    eTuesday    = 1,
    eWednesday  = 2,
    eThursday   = 3,
    eFriday     = 4
};

const int EDaySize = 5;

string EDayName( int iIndex );

// ETripTimeType Enum

enum ETripTimeType
{
    eJourney    = 0,
    eTransit    = 1,
    eWait       = 2
};

const int ETripTimeTypeSize = 3;

string ETripTimeTypeName( int iIndex );

// ENearFar Enum

enum ENearFar
{
    eNear   = 0,
    eFar    = 1
};

const int ENearFarSize = 2;

string ENearFarName( int iIndex );
```

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
 * utility.cc
 * Copyright (C) Matthew Bradley 2010 <mjbradley1971@gmail.com>
 *
 * utility.cc is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * utility.cc is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "utility.h"

#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <cmath>

void Split( string iStr, char iSep, vector<string> &oList, bool discard_empties )
{
    vector<string>    result;
    int p = 0, sp = 0;
    iStr += space;
    do
    {
        // get sub-string
        while ( p != iStr.length() && iStr[p] != iSep )
            p++;
        int pa = sp, pb = p;

        // trim front of white space
        while ( pa < pb && isspace( iStr[pa] ) )
            pa++;

        // trim back of white space
        while ( pa < pb && isspace( iStr[pb-1] ) )
            pb--;

        // add non-empty sub-strings to output list
        if( !(iStr.substr( pa, pb - pa ) == "" && discard_empties) )
            result.push_back( iStr.substr( pa, pb - pa ) );

        sp = ++p;

    } while ( p < iStr.length() );
    if ( result.size() == 1 && result.front().length() == 0 )
        result.clear();
    oList = result;
}

vector<string> GetLines( string iLines )
{
    vector<string> lines;
    Split( iLines, '\n', lines, true );
    return lines;
}

vector<string> GetCells( string iCells )
{
    vector<string> cells;
    Split( iCells, '\t', cells, false );
    return cells;
}
```

```
string RecombineLines( vector<string> iLines )
{
    string recomibined_lines;

    for_all( vector<string>, iLines, line )
    {
        vector<string>::iterator last_iter = iLines.end(); last_iter--;

        recomibined_lines += *line + ( line != last_iter ? "\n" : "" );
    }

    return recomibined_lines;
}

string RecombineCells( vector<string> iCells )
{
    string recomibined_cells;

    for_all( vector<string>, iCells, cell )
    {
        vector<string>::iterator last_iter = iCells.end(); last_iter--;

        recomibined_cells += *cell + ( cell != last_iter ? "\t" : "" );
    }

    return recomibined_cells;
}

int ToInt( string iString )
{
    int l;
    istringstream iss(iString);
    iss >> l;
    return l;
}

float ToFloat( string iString )
{
    float d;
    istringstream iss(iString);
    iss >> d;
    return d;
}

const string k_file_path = "../data/";

string ReadFile( const string iFileName )
{
    FILE *file = fopen( (k_file_path+iFileName).c_str(), "r+" );

    if ( file == NULL )
    {
        clog << "unable to read file " << iFileName << endl;
        return string();
    }

    fseek( file, 0, SEEK_END );
    int length = ftell(file);
    char* buffer = (char *) malloc( length );
    rewind(file);
    fread( buffer, length, 1, file );
    fclose( file );

    string the_return = string( buffer, length );

    free( buffer );

    return the_return;
}
```

```
void WriteFile( const string iFileName, const string iData )
{
    FILE    *file;

    const char*    c_string = iData.c_str();

    file = fopen( (k_file_path+iFileName).c_str(), "w+" );
    fwrite( c_string, iData.length(), 1, file );
    fclose( file );
}

bool IsOdd( int iValue )
{
    return !IsEven(iValue);
}

bool IsEven( int iValue )
{
    return ( ( floor(float(iValue)/2.0) * 2 ) == iValue );
}

bool IsWhole( float iValue )
{
    return iValue == lroundf(iValue);
}

float RandomLevel()
{
    return float(rand())/float(RAND_MAX);
}

int RandomChoice( int iSize )
{
    int choice = floor( RandomLevel() * float(iSize) );

    // boundry case
    if( choice == iSize )
        choice = iSize-1;

    return choice;
}

string BoolToString( bool b )
{
    return ( b ? "true" : "false" );
}

bool StringToBool( string s )
{
    return ( s == "true" ? true : false );
}

string EDirectionName( int iIndex )
{
    string the_name[2];

    the_name[0] = "inbound";
    the_name[1] = "outbound";

    return the_name[iIndex];
}

string EDirectionExtendedName( int iIndex )
{
    string the_name[3];

    the_name[0] = "inbound";
    the_name[1] = "outbound";
    the_name[2] = "eitherdirection";

    return the_name[iIndex];
}
```

```
string EFirstLastName( int iIndex )
{
    string the_name[2];

    the_name[0] = "first";
    the_name[1] = "last";

    return the_name[iIndex];
}

string EArriveDepartName( int iIndex )
{
    string the_name[2];

    the_name[0] = "arrive";
    the_name[1] = "depart";

    return the_name[iIndex];
}

string ETemporalTypeName( int iIndex )
{
    string the_name[2];

    the_name[0] = "before";
    the_name[1] = "after";

    return the_name[iIndex];
}

string EMinMaxName( int iIndex )
{
    string the_name[2];

    the_name[0] = "min";
    the_name[1] = "max";

    return the_name[iIndex];
}

string EFromToName( int iIndex )
{
    string the_name[2];

    the_name[0] = "from";
    the_name[1] = "to";

    return the_name[iIndex];
}

string EEntryExitName( int iIndex )
{
    string the_name[2];

    the_name[0] = "entry";
    the_name[1] = "exit";

    return the_name[iIndex];
}

string EBoardAlightName( int iIndex )
{
    string the_name[2];

    the_name[0] = "board";
    the_name[1] = "alight";

    return the_name[iIndex];
}

string EPosNegName( int iIndex )
{
    string the_name[2];

    the_name[0] = "positive";
    the_name[1] = "negative";

    return the_name[iIndex];
}
```

```
string EInnerOuterName( int iIndex )
{
    string the_name[2];

    the_name[0] = "inner";
    the_name[1] = "outer";

    return the_name[iIndex];
}

string EDayName( int iIndex )
{
    string the_name[5];

    the_name[0] = "monday";
    the_name[1] = "tuesday";
    the_name[2] = "wednesday";
    the_name[3] = "thursday";
    the_name[4] = "friday";

    return the_name[iIndex];
}

string ETripTimeTypeName( int iIndex )
{
    string the_name[3];

    the_name[0] = "journey";
    the_name[1] = "transit";
    the_name[2] = "wait";

    return the_name[iIndex];
}

string ENearFarName( int iIndex )
{
    string the_name[2];

    the_name[0] = "near";
    the_name[1] = "far";

    return the_name[iIndex];
}
```

< blank page >

< blank page >

< blank page >

< blank page >

< blank page >

Appendix Two

Output Ultra-efficient Timetable

Notes

1. This timetable only shows the revenue buses services. It does not show the deadhead bus runs.

```
04:30 - 04:45 inbound

Services per
15-minutes        Stopping Pattern

3           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


04:45 - 05:00 inbound

Services per
15-minutes        Stopping Pattern

3           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


05:00 - 05:15 inbound

Services per
15-minutes        Stopping Pattern

3           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


05:15 - 05:30 inbound

Services per
15-minutes        Stopping Pattern

4           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


05:30 - 05:45 inbound

Services per
15-minutes        Stopping Pattern

4           17 -- 15 -- 13 -- -- -- - - - - 5 4 - - - 0
3           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


05:45 - 06:00 inbound

Services per
15-minutes        Stopping Pattern

3           17 -- 15 -- -- -- -- -- - - - - - - - - - 0
3           17 16 15 -- 13 -- -- -- - 8 - - 5 4 - - - 0
3           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

06:00 - 06:15 inbound

Services per 15-minutes	Stopping Pattern
4	17 16 -- -- -- -- -- -- - - - - - - - - - 0
4	17 -- 15 -- 13 -- -- 10 - 8 - - - - - 2 - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

06:15 - 06:30 inbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 12 -- -- - - - - - - - - 0
5	17 -- -- -- -- -- -- -- - - - - 5 - - - - 0
5	17 16 15 -- 13 -- -- 10 - - - 6 - 4 - 2 - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

06:30 - 06:45 inbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- 10 - - - 6 - - - - - 0
6	17 16 -- -- -- -- -- -- - - - - - - 3 - - 0
5	17 16 15 -- 13 -- -- -- 9 - - - 5 4 - 2 - 0
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

06:45 - 07:00 inbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- 10 - - - - - - - - - 0
6	17 16 15 -- -- -- -- -- - - - - - - - - - 0
5	17 -- -- -- 13 -- -- -- - - - 6 5 4 3 2 - 0
6	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

07:00 - 07:15 inbound

Services per 15-minutes	Stopping Pattern
5	-- -- -- -- 13 12 -- 10 - - - - - - - - - 0
6	17 16 -- 14 -- -- -- -- - - - - - - - - - 0
6	17 -- 15 -- 13 -- 11 -- - - - - 5 4 3 2 - 0
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

07:15 - 07:30 inbound

Services per 15-minutes	Stopping Pattern
5	-- -- -- -- 13 12 -- -- - - - - - - - - 0
7	17 16 -- -- -- -- -- -- - - - - 5 - - - - 0
6	17 -- 15 -- 13 -- -- -- - - - - - 4 3 2 - 0
6	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

07:30 - 07:45 inbound

Services per 15-minutes	Stopping Pattern
6	-- -- -- -- 13 12 -- 10 - - - - - - - - - 0
6	17 -- -- 14 -- -- -- -- - - - - - - - - - 0
6	17 16 15 -- 13 -- 11 -- - - - - - 4 3 - - 0
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

07:45 - 08:00 inbound

Services per 15-minutes	Stopping Pattern
5	-- -- -- -- 13 -- -- 10 - - - - - - - - - 0
5	17 16 -- -- -- -- -- -- - - - - - - - - - 0
5	17 -- 15 -- 13 12 -- -- - - - - 5 4 3 2 - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

08:00 - 08:15 inbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 12 -- -- - - - - - - - - - 0
4	17 -- -- 14 -- -- -- 10 - - - - - - - - - 0
4	17 16 15 -- 13 -- -- -- 9 - - - - 4 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

08:15 - 08:30 inbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- 11 -- - - - - - - - - - 0
5	17 16 15 -- -- -- -- 10 9 - 7 - - - - - - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

08:30 - 08:45 inbound

Services per 15-minutes	Stopping Pattern
4	17 16 15 -- -- -- -- 10 - - - - - - - - - 0
3	17 -- 15 -- 13 -- -- -- - - - - - - 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

08:45 - 09:00 inbound

Services per 15-minutes	Stopping Pattern
3	17 -- 15 -- -- -- -- -- - - - - - - - - - 0
3	17 16 15 -- 13 -- -- 10 - - - - - - - 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

09:00 - 09:15 inbound

Services per 15-minutes	Stopping Pattern
4	17 -- 15 -- 13 -- -- -- - - - - - - - - - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

09:15 - 09:30 inbound

Services per 15-minutes	Stopping Pattern
4	17 16 15 -- 13 -- -- 10 - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

09:30 - 09:45 inbound

Services per 15-minutes	Stopping Pattern
4	17 -- 15 -- 13 -- -- 10 - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

09:45 - 10:00 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

10:00 - 10:15 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 -- -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

10:15 - 10:30 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

10:30 - 10:45 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

10:45 - 11:00 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

```
11:00 - 11:15 inbound

Services per
15-minutes        Stopping Pattern

5           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


11:15 - 11:30 inbound

Services per
15-minutes        Stopping Pattern

4           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


11:30 - 11:45 inbound

Services per
15-minutes        Stopping Pattern

4           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


11:45 - 12:00 inbound

Services per
15-minutes        Stopping Pattern

4           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


12:00 - 12:15 inbound

Services per
15-minutes        Stopping Pattern

5           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


12:15 - 12:30 inbound

Services per
15-minutes        Stopping Pattern

5           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


12:30 - 12:45 inbound

Services per
15-minutes        Stopping Pattern

4           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0


12:45 - 13:00 inbound

Services per
15-minutes        Stopping Pattern

4           17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

13:00 - 13:15 inbound

Services per 15-minutes	Stopping Pattern
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

13:15 - 13:30 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

13:30 - 13:45 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

13:45 - 14:00 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

14:00 - 14:15 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

14:15 - 14:30 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

14:30 - 14:45 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

14:45 - 15:00 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

15:00 - 15:15 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

15:15 - 15:30 inbound

Services per 15-minutes	Stopping Pattern
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

15:30 - 15:45 inbound

Services per 15-minutes	Stopping Pattern
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

15:45 - 16:00 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

16:00 - 16:15 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

16:15 - 16:30 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

16:30 - 16:45 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

16:45 - 17:00 inbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 12 -- -- - - - - 5 4 - 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

17:00 - 17:15 inbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- -- -- - - - 6 5 4 - - - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

17:15 - 17:30 inbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- 11 -- - - - 6 5 4 3 - - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

17:30 - 17:45 inbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- -- 9 8 - 6 5 - 3 - - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

17:45 - 18:00 inbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- -- 9 - - - 5 - 3 2 - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18:00 - 18:15 inbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- -- 9 - - 6 5 - 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18:15 - 18:30 inbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- -- 10 9 - - - 5 - 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18:30 - 18:45 inbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18:45 - 19:00 inbound

Services per 15-minutes	Stopping Pattern
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

19:00 - 19:15 inbound

Services per 15-minutes	Stopping Pattern
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

19:15 - 19:30 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

19:30 - 19:45 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

19:45 - 20:00 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

20:00 - 20:15 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

20:15 - 20:30 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

20:30 - 20:45 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

20:45 - 21:00 inbound

Services per
15-minutes Stopping Pattern

3 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

21:00 - 21:15 inbound

Services per
15-minutes Stopping Pattern

3 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

21:15 - 21:30 inbound

Services per
15-minutes Stopping Pattern

3 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

21:30 - 21:45 inbound

Services per
15-minutes Stopping Pattern

3 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

21:45 - 22:00 inbound

Services per
15-minutes Stopping Pattern

3 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

22:00 - 22:15 inbound

Services per
15-minutes Stopping Pattern

3 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

22:15 - 22:30 inbound

Services per
15-minutes Stopping Pattern

3 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

22:30 - 22:45 inbound

Services per
15-minutes Stopping Pattern

3 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

22:45 - 23:00 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

23:00 - 23:15 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

23:15 - 23:30 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

23:30 - 23:45 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

23:45 - 24:00 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

24:00 - 24:15 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

24:15 - 24:30 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

24:30 - 24:45 inbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

04:15 - 04:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

04:30 - 04:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

04:45 - 05:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

05:00 - 05:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

05:15 - 05:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

05:30 - 05:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

05:45 - 06:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

06:00 - 06:15 outbound

Services per 15-minutes	Stopping Pattern
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

06:15 - 06:30 outbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

06:30 - 06:45 outbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- -- - - - 6 5 4 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

06:45 - 07:00 outbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- -- -- - - - 6 5 - - - - 0
3	-- -- -- -- 13 -- -- -- 9 - - - - 4 - 2 1 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

07:00 - 07:15 outbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- 9 - - - 5 4 - - - 0
3	-- -- -- -- 13 -- 11 -- - - - 6 - - 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

07:15 - 07:30 outbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- -- - - 7 - - - 3 2 - 0
3	-- -- -- -- 13 -- -- 10 - 8 - 6 - 4 - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

07:30 - 07:45 outbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- -- 9 - - - 5 - 3 - - 0
4	-- -- -- -- 13 -- -- 10 - 8 7 6 - - - 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

07:45 - 08:00 outbound

Services per 15-minutes	Stopping Pattern
5	-- -- -- -- 13 -- -- -- 9 - - - - - 3 2 - 0
4	-- -- -- -- 13 -- -- -- - 8 7 - 5 4 - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

08:00 - 08:15 outbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- -- - - - - - - 3 2 - 0
4	-- -- -- -- 13 -- -- -- - 8 7 6 5 4 - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

08:15 - 08:30 outbound

Services per 15-minutes	Stopping Pattern
5	-- -- -- -- 13 -- -- 10 - - - - 5 4 3 2 - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

08:30 - 08:45 outbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- -- 9 - - - 5 - 3 2 - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

08:45 - 09:00 outbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- -- - - - 6 5 4 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

09:00 - 09:15 outbound

Services per 15-minutes	Stopping Pattern
4	-- -- -- -- 13 -- -- -- 9 8 - - 5 - 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

09:15 - 09:30 outbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 12 -- -- 9 - - - 5 - 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

09:30 - 09:45 outbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- -- -- 9 - - - 5 - 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

09:45 - 10:00 outbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- -- -- 9 - - - 5 - 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

10:00 - 10:15 outbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- 11 -- - - - - 5 - 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

10:15 - 10:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- -- -- 13 -- -- -- - 8 - - - - - 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

10:30 - 10:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 -- -- 13 -- -- -- - - - - - - 3 - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

10:45 - 11:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- 15 -- 13 -- -- -- - - - - - - - 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

11:00 - 11:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 -- -- 13 -- -- -- - - - - - - 3 - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

11:15 - 11:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- -- -- 13 -- -- -- - - - - 5 - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

11:30 - 11:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

11:45 - 12:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 -- -- 13 -- -- 10 - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

12:00 - 12:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 -- -- 13 -- -- 10 - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

12:15 - 12:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 -- -- 13 -- -- -- - - - 6 - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

12:30 - 12:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 -- -- 13 -- -- -- - 8 - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

12:45 - 13:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

13:00 - 13:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

13:15 - 13:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 -- -- 13 -- -- -- - - - - 5 - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

13:30 - 13:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- -- -- 13 -- -- -- - 8 - - 5 - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

13:45 - 14:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- 15 -- 13 -- -- -- - - - - 5 - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

14:00 - 14:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

14:15 - 14:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- 13 -- -- 10 - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

14:30 - 14:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

14:45 - 15:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

15:00 - 15:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

15:15 - 15:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- -- -- 13 -- -- 10 - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

15:30 - 15:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 -- -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

15:45 - 16:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

16:00 - 16:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

16:15 - 16:30 outbound

Services per 15-minutes	Stopping Pattern
4	17 -- 15 -- 13 -- -- 10 - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

16:30 - 16:45 outbound

Services per 15-minutes	Stopping Pattern
4	17 16 15 14 13 -- -- 10 - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

16:45 - 17:00 outbound

Services per 15-minutes	Stopping Pattern
4	17 -- 15 -- 13 -- -- -- - - - - - - - - - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

17:00 - 17:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- 15 -- -- -- -- -- - - - - - - - - - 0
3	17 16 -- -- 13 -- -- 10 - 8 - - 5 - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

17:15 - 17:30 outbound

Services per 15-minutes	Stopping Pattern
4	17 -- -- -- 13 -- -- -- - - - - - - - - - 0
4	17 16 15 -- 13 -- 11 10 - - - 6 5 4 - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

17:30 - 17:45 outbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 12 -- -- - - - - - - - - - 0
4	17 16 15 14 -- -- -- -- - - - - - - - - - 0
3	17 -- 15 -- 13 -- -- 10 - - - - 5 4 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

17:45 - 18:00 outbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- -- -- - - - - - - - - - 0
4	17 16 -- 14 -- -- -- -- - - - - - - - - - 0
4	17 -- 15 -- 13 -- 11 -- - - - - 5 4 3 2 - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18:00 - 18:15 outbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- -- -- - - - - - - - - - 0
5	17 16 -- -- -- -- -- 10 - - - - - - - - - 0
4	17 16 15 -- 13 -- -- -- - - - 6 - 4 3 2 - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18:15 - 18:30 outbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- -- -- - - - - - - - - - 0
5	17 16 -- -- -- -- -- -- - - - - - - - - - 0
5	17 -- 15 -- 13 -- -- 10 - - - 6 5 - 3 2 - 0
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18:30 - 18:45 outbound

Services per 15-minutes	Stopping Pattern
6	17 16 15 -- -- -- -- -- - - - - 5 - - - - 0
5	17 -- -- -- 13 12 -- -- - - - - - - 3 2 - 0
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18:45 - 19:00 outbound

Services per 15-minutes	Stopping Pattern
5	17 16 -- -- -- -- -- -- - - - 6 - - - - - 0
5	17 -- -- -- 13 12 -- -- - 8 - - 5 - - 2 - 0
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

19:00 - 19:15 outbound

Services per 15-minutes	Stopping Pattern
3	-- -- -- -- 13 -- -- -- - - - - - - - - - 0
4	17 -- -- -- -- -- -- -- - - - - - - - - - 0
4	17 16 15 -- 13 -- -- -- - - - - 5 4 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

19:15 - 19:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- -- -- -- -- -- -- - - - - - - - - - 0
3	17 -- 15 -- 13 -- -- -- - - - - - - - - - 0
3	17 16 -- -- 13 -- -- 10 - - - - 5 - 3 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

19:30 - 19:45 outbound

Services per 15-minutes	Stopping Pattern
4	17 16 -- -- -- -- -- -- - - - - - - - - - 0
4	17 -- 15 14 13 -- -- -- - - - - 5 - - 2 - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

19:45 - 20:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- -- -- -- -- -- -- - - - - - - - - - 0
3	17 16 15 -- 13 -- -- -- - - - - - 4 - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

20:00 - 20:15 outbound

Services per 15-minutes	Stopping Pattern
4	17 -- -- 14 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

20:15 - 20:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 -- -- 12 -- 10 - - - - 5 - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

20:30 - 20:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- 15 -- -- -- -- 10 - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

20:45 - 21:00 outbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

21:00 - 21:15 outbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

21:15 - 21:30 outbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

21:30 - 21:45 outbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

21:45 - 22:00 outbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

22:00 - 22:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 -- -- -- 13 -- -- -- - - - - - - - - - 0
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

22:15 - 22:30 outbound

Services per 15-minutes	Stopping Pattern
5	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

22:30 - 22:45 outbound

Services per 15-minutes	Stopping Pattern
4	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

22:45 - 23:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

23:00 - 23:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

23:15 - 23:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

23:30 - 23:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

23:45 - 24:00 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

24:00 - 24:15 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

24:15 - 24:30 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

24:30 - 24:45 outbound

Services per 15-minutes	Stopping Pattern
3	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0