Department of Computing

Applications of Mathematical Optimisation to Non-Functional Requirements in Software Engineering

Amy Affleck

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

September 2016

Declaration

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgment has been made. This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Amy Affleck Digitally signed by Amy Affleck Date: 2017.11.10 12:50:47 +08'00' 08/08/2017

Amy Affleck Date

Abstract

Non-functional requirements are the least addressed element in software development; their neglect is a leading contributor to project failure. Historically, and frequently, functional requirements are considered the driving force of project development, often to the exclusion of their non-functional counterparts. However, since the turn of the century, numerous frameworks, methods, and paradigms have emerged that highlight the effect of non-functional requirements and actively address their involvement in project design. While diverse, these approaches are either limited to mere representation, or typically boast qualitative evaluation methods; current research has taken the direction of developing quantitative evaluation algorithms.

The most prominent paradigm to develop due to this shifting focus is Goal-Oriented Requirements Engineering (GORE), an approach that graphically models all system aspects as goals to be achieved, regardless of their classification as functional or non-functional. Within this paradigm, the work presented here is focused on the Non-Functional Requirements (NFR) Framework and the Goal-Oriented Requirements Language (GRL).

The Non-Functional Requirements Framework is a qualitative method that bridges the gap between the idea of non-functional requirements and a software design that encompasses these ideas. The framework functions by modelling non-functional requirements and the associated implementation methods, eventually resulting in a fully documented decision regarding the implementation of said methods. The Goal-Oriented Requirements Language takes an alternative approach by providing modelling elements rather than specifying processes. This allows the language the versatility to implement a range of frameworks.

Current work within this field focuses on the development and evolution of evaluation algorithms and modelling applications, only a handful of work has been completed in developing algorithms and techniques that can *actively* assist in the decision making process. Such algorithms would be monumental in overcoming the scalability issues inherent in developers visually assessing large graphical models.

This thesis investigates the development of decision algorithms through the use of mathematical optimisation, specifically Mixed Integer Linear Programming (MILP), presenting alternatives for both the NFR Framework and GRL. Key factors in the viability assessment of the proposed optimisation models include scalability, consistency, and automation.

Acknowledgements

When I first started work on my PhD I had very little idea about what the journey would involve, or the hurdles that would appear. I set out with a love of learning, knowledge, problem solving, and a little naivety. Thankfully, I was surrounded by amazing colleges, friends, and family who have, over the years, all helped me achieve my goal.

Firstly, I must thank my supervisors, Aneesh Krishna and Narasimaha R Achuthan (Archie). Since I was an undergraduate student enrolled in one of his units Aneesh has never failed to show the utmost confidence in my abilities, even when my own self-doubt was my largest obstacle. He provided guidance to this new world I was trying to enter, and was always understanding and patient. Coming from a software engineering background and taking on an almost cross-discipline goal was, initially, extremely daunting. As such, I was extremely grateful for Archie's guidance and patience, not only while I was trying to understand the maths I was applying but also the correct format and presentation. Over time I grew more confident with the mathematical part of my work at which point I became extremely thankful to have a supervisor who was willing to be a sounding board for my ideas, and could, at times, point out where I had failed to consider all possibilities. Although, I will never forget the first time I solved a problem he did not know the answer to, I believe that was the moment I realised what it truly meant to undertake a PhD.

My supervisors were amazing, but on a day-to-day basis it was my fellow research students who created my work environment. I would like to thank my officemates, Arlen, Ming, Ross, and Stefan, for creating a positive, friendly, and enjoyable environment. I'd especially like to thank Arlen and Stefan for being my "coffee-buddies", though I may have over used them at times. I would also like to thank the other students and staff members, they created an equally amazing department that was a pleasure to be a part of, specifically I would like to thank Patrick, Dave, Hannes, Mark, and Mike, they provided help when I was first starting, stimulating conversation when I could no longer sit still, and their support and confidence has always been greatly appreciated.

I would also like to thank my mother and sister for their love and patient support over the years. Especially my big sister, who without fail, will not let me go more than a couple of weeks without checking in or catching up.

Finally, I would like to thank my housemate Katie, over the years she has put up with my odd hours and behaviour, including moving my entire study space to the lounge room in order to meet a deadline. She has been a wonderful constant in my life and I am forever thankful for her amazing grammar skills.

Publications

During the course of this research the following publications were produced:

Paper:	Supporting Quantitative Reasoning of Non-Functional Requirements: A Process-Oriented Approach.
Authors:	Amy Affleck and Aneesh Krishna
Publication:	International Conference on Software and System Process (ICSSP), 2012
Paper:	Optimal Selection of Operationalizations for Non-Functional Requirements.
Authors:	Amy Affleck, Aneesh Krishna, and Narasimaha R. Achuthan
Publication:	Ninth Asia-Pacific Conference on Conceptual Modelling (APCCM), 2013
Paper:	Non-Functional Requirements Framework: A Mathematical Programming Approach.
Authors:	Amy Affleck, Aneesh Krishna, and Narasimaha R. Achuthan
Publication:	The Computer Journal, Vol.58 2015

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software Engineering was originally defined by [1] as the establishment and use of engineering principles in order to obtain reliable and efficient software. The greatest obstacle to this goal is project failure. Project failure occurs when projects exceed the allocated budget, run over time, or lack required functionality. Research has shown that the leading cause of project failure is the inadequate handling of system requirements, that is the needs that the system must fulfil [2, 3, 4, 5, 6].

The assessment, definition, and justification of requirements falls into the domain of *Requirements Engineering* [7, 8, 9]; formally defined by [10] as:

> "...the branch of software engineering that is concerned with the real world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families."

The early phases of requirements engineering can be defined by the following processes: *eliciting requirements*, *modelling and analysing requirements*, *communicating requirements*, *agreeing requirements*, and *evolving requirements* [11]. Requirements fall into two categories; functional, and non-functional. Functional requirements define the tasks a system must be capable of completing. Historically, they have been the core concern for developers, and the driving force of project development. Conversely, non-functional requirements are concerned with *how* the system must behave while completing the specified tasks; as their criteria is less defined than their functional counterparts, they receive considerably less attention. The negligence of non-functional requirements *continues* to be a leading cause of project

failure [12, 13].

Throughout the history of software engineering, the expression of non-functional requirements has evolved from mere natural language expression to methods including *Unified modelling Language (UML)* extensions [14, 15, 16], taxonomy based specification, and their express inclusion in functional requirements [17]. These methods are based on incorporating non-functional requirements into existing specifications, a novel approach in this regard is *Goal-Orientated Requirements Engineering (GORE)* [18, 11]. GORE is a form of conceptual modelling that treats requirements as goals to be achieved, regardless of their status as functional or non-functional [19, 20, 21, 22]. The ability to model both functional and non-functional requirements, their interactions, and the predominantly graphical specification, are key factors in the growing popularity of GORE. There are several renowned frameworks in the GORE methodology including the *i* Framework* [23], *Knowledge Acquisition in Automated Specification (KAOS)* [24], *Non-Functional Requirements (NFR) Framework* [25], *Tropos* [26], and Techne [27].

While the focus of these frameworks initially relied on qualitative reasoning, a significant amount of work has been done in developing quantitative alternatives, several of these alternatives are discussed in chapter 2. The common factor shared by current evaluation techniques is heavy reliance on the developers ability to simultaneously consider an *entire* graph and all its interdependencies; the difficulty of this evaluation increases exponentially as the size of the graph increases. After careful consideration, developers decide between implementation alternatives and the *results* of the decisions propagated in order to view the consequences. While considerable attention has been applied to *how* the results are propagated, significantly less attention has been given to the decision process itself. Hence, the work presented in this thesis is focused on developing suitable techniques that do not suffer from scalability issues, and ensure the best decisions are reached. As such the following research objectives are defined.

1.1 Research Objectives

Objective 1: Decision Aid

> *Can evaluation methods be developed that accurately* recommend *design decisions rather than* assess *their effectiveness?*

Justification: Value propagation techniques are abundant in current research, techniques designed to determine the best decision are however, lacking. This presents a unique area that the work in this thesis aims to explore.

Objective 2: Overcome Scalability Issues and Computation Time

> *Can the proposed methods be applied to graphs of considerable size without suffering degradation, either in their accuracy or computation time?*

Justification: Any evaluation technique must be capable of producing a result within a reasonable time period regardless of graph size. The application and usage of any technique failing this objective would be infeasible.

Objective 3: Produce an Optimal Result

> *Do the proposed methods confidently and consistently produce results that cannot be improved?*

Justification: Any automated decision tool must be able to produce the best result or risk being redundant. By ensuring that no better result exists, the developed technique can be considered accurate and trustworthy.

Objective 4: Automation

Can the proposed methods be calculated with minimal developer input?

Justification: Some degree of developer interaction is favourable as it allows for customisation and adjustment of the results. A decision aid that requires constant developer input would fail in meeting it intended purpose as the developer would be able to come to the same decisions on their own.

1.2 Thesis Structure

This thesis has been organised into 6 additional chapters; an outline of each of these chapters is provided below.

Chapter 2 provides relevant background information, this includes an overview of techniques that acknowledge non-functional requirements and quantitative evaluation techniques. The difference between evaluation and automation also receives special attention, as does the current standing of operations research within goal modelling. The remaining portion of the chapter discusses the application of mathematical optimisation to GORE models. This includes establishing the terminology used in later chapters, and the process by which the optimisation schemas should be interpreted and applied. Additionally, the chapter discusses the benefits of applying mathematical optimisation and its use in conflict resolution.

Chapter 3 applies the concept of mathematical optimisation to the Non-Functional Requirements Framework by developing an optimisation schema capable of evaluating graphs constructed using the framework. Additionally, the chapter develops an innovative approach to sensitivity analysis that provides results applicable to conflict resolution. Both these methods are applied to several exemplars and test cases, the results are then analysed to determine accuracy and scalability. The results are also compared to existing quantitative decision making methods. Finally, the chapter analyses the computation time

of the individual optimisation process and the sensitivity analysis in order to determine the feasibility of the approaches.

Chapter 4 develops a comprehensive optimisation schema for the Goal-Orientated Requirements Language. The developed schema includes several different evaluation strategies, each of which is applied to several test cases. The results of the test cases are analysed to determine their effectiveness, and the computation time is examined to determine the viability of the optimisation.

Chapter 5 develops an alternative approach to GRL. This alternative is based on the idea of an evidence based evaluation, by incorporating this approach in the initial optimisation schema the time complexity of produced models is greatly decreased. The test cases from the previous chapter are used to verify this optimisation schema, the results are analysed and compared to those of the previous chapter. Once again the time complexity of the optimisation schema is evaluated to determine viability.

Chapter 6 provides a summary of the work presented in this thesis, an assessment of the research objectives, and outlines potential areas of future work.

Chapter 2

Background and Related Work

The work presented in this thesis falls into the domain of Goal-Oriented Requirements Engineering (GORE), it also utilises techniques associated with *operations research (OR)*. This chapter establishes the relevant background and current standing of GORE, and explores the application of operations research within the GORE paradigm.

Section 2.1 introduces the usage of quantitative values in requirements engineering, section 2.2 then discusses several alternative GORE frameworks and how they have utilised quantitative values. Section 2.3 discusses the use of goal models outside their association with specific frameworks, focusing instead on the *concept* of goals, this discussion includes the Goal-Oriented Requirements Language. Section 2.4 addresses the issue of evaluation based algorithms compared to decision making and automation. Section 2.5 addresses operations research, exploring its applications to GORE, focussing on mathematical optimisation.

Section 2.6.1 provides a brief introduction to mathematical optimisation, establishes the associated terminology, and elucidates mutual exclusion in a linear system. Section 2.6.2 briefly outlines the benefits of applying optimisation to goal models and its potential use in conflict resolution. Finally, section 2.6.3 explains how the optimisation schemas in future chapters shall be presented, and provides a small worked example to demonstrate how the schemas should be interpreted and implemented.

2.1 Quantitative Values in Requirements Engineering

Traditionally, goal model evaluation has been driven by qualitative values, a practise that still holds considerable influence due to the, at times, uncertain, informal, or controversial nature of non-functional requirements. Evolution of the methodologies, frameworks, and approaches used to model non-functional requirements have, over time, removed much of this uncertainty. Concomitantly, significant research has been done into extracting and using quantitative values, related to both non-functional requirements and goal models as a whole.

A key examination of qualitative verse quantitative goal model labels was presented by [28] at the 6th i* workshop [29]. The work discusses various adaptations of quantitative values, from representations of labels, to those with their own distinct semantics. Through work with several focus groups it was determined that quantitative values can improve understandability. Several approaches towards quantitative value selection are identified and compared in [30]. The work uses GRL goal models, in two government test cases, to examine group decision approaches, and determine the feasibility of the selected approaches.

2.2 Goal Model Frameworks

Goal models are frequently used in software engineering to graphically represent the requirements, processes, and behaviours of a software system; since the concept of goal modelling emerged, several frameworks have been developed to take advantage of its principles. Prominently, the NFR Framework, i*, KAOS, Tropos, and Techne.

2.2.1 NFR Framework

The NFR Framework models non-functional requirements as *softgoals*[1], identifies methods of achieving them (*operationalizations*), and qualitatively weighs the relationships. Once the SIG has been constructed, developers examine the model and decide which implementation methods should be accepted, the results of these decisions are propagated through the remaining graph in order to determine qualitative labels for all softgoals. One of the major benefits of the NFR Framework is the complete documentation of every step taken by developers.

Quantitative evaluation based on parent level softgoals is explored by [31], proposing a mathematical formalisation for the approach specified by [32]. The work formalises the concept of a softgoal weight, based on the type of connected children, associated with this is a method of propagation. This allows the effect of different operationalization selections to be calculated and compared.

Previous work introduced the idea of extending the NFR Framework to support quantitative reasoning; including the quantitative propagation of decisions [33].The format of this extension follows the original framework but uses quantitative priority and relationship values. These values are used to determine which operationalizations have the greatest benefit on the model, the operationalizations are then accepted and again the values propagated to the remainder of the graph.

2.2.2 i* Framework

The i* Framework is a qualitative framework applied during the early phase of requirements engineering in order to gain an understanding of the problem domain. The models created by the i* Framework outline various softgoal, goal, task, and resource dependencies between actors; models can also outline the rationale behind

[1]The term softgoal was coined in [21] to be *satisficed* in a *Softgoal Interdependency Graph (SIG)* [25]. The framework methodically refines the softgoals and has since become a standard term when modelling objectives without explicit satisfaction criteria; most commonly non-functional requirements.

individual actors. The idea of the i* Framework is to model the entire process of the system, not just the software components; once the processes are understood the model is then used to elicit requirements. The i* Framework is increasingly popular, with the 9th annual workshop being held in 2016 presenting several ideas for varying applications and extensions [34].

2.2.3 Knowledge Acquisition in Automated Specification

The KAOS model is a qualitative approach to requirements elicitation that extracts requirements from goal diagrams. These goals can contain a combination of both functional and non-functional system aspects; the models include methods of achieving these goals. KAOS is purely concerned with the goals of the system as opposed to i* which focuses on actors and their interactions. Additionally, KAOS is oriented towards goal satisfaction whereas i* is more oriented towards goal satisficing.

A quantitative extension to the KAOS framework that includes partial goal satisfaction and probabilistic theory is developed by [35]. The main goal of this work was to overcome the limitations of conclusions and accuracy in the qualitative framework. The work includes the presentation of a systematic method for elaborating goal models based on the developed techniques.

The author of [36] applies the qualitative propagation algorithm utilised by the NFR Framework — first introduced in [37] — to the KAOS Framework before developing a lightweight quantitative alternative based on comparative values. The initial approach is improved by the introduction of gauge values which are derived from the requirements specification.

2.2.4 Tropos

Similar to the i* Framework, Tropos [26] is based on the idea of agents and goals. However, the approach differs as Tropos presents a *requirements-driven* method

of software development. The concepts used are not limited to requirements engineering, instead persisting through the entire software life-cycle. As with the i* Framework and KAOS, Tropos presents a qualitative approach.

2.2.5 Techne

A different adaptation of goal modelling is presented by Techne [27], an *abstract language* used in the early stages of requirements engineering. Techne's graph structures offer an expressive way to model mandatory and optional requirements, preferences, domain assumptions, and arguments for or against any part of a requirements model. However, Techne is strictly qualitative and does not accommodate precedence links.

2.3 Framework Independent Goal Model Usage

The idea of using goal models to represent the system holds such appeal that a significant portion of recent work subscribes to the notion of goal models, without enforcing the rigidity of a specific framework. Often, evaluation techniques developed in such a manner can be easily adapted to a number of frameworks. The *Goal-Oriented Requirements Language (GRL)* is the manifestation of framework independent goal model usage [38]. The *language* presents an assortment of node and relationship classifications that can be combined to model specific frameworks or merely express developer thoughts and understandings. The popularity and versatility of GRL has led to it being included in the *User Requirements Notation (URN)*, part of the *ITU-T* — Telecommunication Standards, recommendation Z series [39].

A formal method for quantitative reasoning with goal models is introduced by [40]. Precise semantics have been specified for all goal relationships based heavily on first order logic and mathematical reasoning. These goal relationships use a probabilistic model, and a label propagation algorithm that is shown to be sound and

complete.

An in-depth approach to goal priorities, creating both optional and mandatory classifications, is introduced by [41]. A quantitative approach is then defined using optional priorities to evaluate design alternatives for the mandatory priorities. This is analogous to the kind of result that popular quantitative priority elicitation techniques produce. The proposed approach is implemented using a preference based planner to create a set of optimal solutions.

Another application of probabilistic theory is presented in [42]. This approach applies probability, and KAOS reminiscent logic in order to prioritise obstacles for countermeasure selection. The aim of the work is to create a robust solution through the application of probability distributions and statistical analysis.

One of the major works in quantitative evaluation for GRL is [43]. The work develops and compares three evaluation methods for GRL, a traditional qualitative approach, a quantitative approach, and a hybrid approach. The work is comprehensive, allowing for various evaluation options, as well as a top-down or bottom-up approach and the proposed methods support all available relationships.

GRL provides the tools necessary to utilise the concepts of goals in accordance to developers requirements, it presents a flexible, versatile solution. However, as it does not present a framework or specific application there are no guidelines or rules to aid developers in making correct decisions.

2.4 Evaluation versus Automation

The frameworks and methods discussed thus far are focused on the evaluation of developer *decisions*, comparatively little research has been conducted in transitioning from an evaluation or analysis paradigm to that of automation or decision *making*.

The work presented by [35] — as discussed above — is extended by [44] to overcome scalability issues by presenting automated techniques for simulating the previous quantitative models and, developing a multi-objective optimisation algorithm to explore the alternative designs. This method applies stochastic simulation in order to evaluate alternatives, though developer intervention is eventually required to compare distinct design choices.

Unique in their application of automation to qualitative model,s [45, 46] present an automatic reasoning mechanism for *qualitative label propagation*. The purpose of the work is to determine if it is possible to completely satisfice a root node under the NFR Framework. This method is explicitly designed for a qualitative evaluation of the NFR Framework, and the presented methodology is reminiscent of the *logic programming paradigm*.

One approach to the issue of automation is the application of a *Boolean Satisfiability Problem (SAT)* as was conducted by [47], the work is limited to AND/OR refinement links and is only demonstrated on safety and liveness goals. While the work does suffer from scalability issues, it does prove that automation can be applied to determine which set of (boolean) agents can achieve the system goals. Additionally, the exclusion of contribution and dependency relationships greatly decreases its applicability to larger scopes.

The application of fuzzy numbers and *Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS)* to create a decision making approach has been explored by [48]. This approach is implemented through two steps, first developers use fuzzy numbers with stakeholder input to determine evaluation criteria, then the scores generated from the evaluation criteria are utilised to determine the best alternative. The proposed alternative must still be specified by the developers and the process was only applied to goal models containing AND/OR relationships; manually generating alternatives in a graph containing contribution style relationships is exponentially more difficult that in AND/OR systems.

Exploration of applying the decision process used by criteria hierarchies to goal models has been explored by [49], given of course that the goal model meets certain

structural characteristics. The work is highly based on preferences and eliciting contribution measures. The approach uses the results of the *analytic hierarchy process* to determine the optimal solution.

Evaluation and automation both have their uses and fill unique niches. Evaluation specifically has historically been a key focal point as it allows developers to determine the consequences of their decisions while still designing a solution and evaluating alternatives. Such evaluation methodologies allow developers to maintain complete control, however, such algorithms are often plagued by scalability issues. If developers must make every decision and examine the results then, naturally, as project size increases the number of decisions must also increase. Automation, on the other hand, aims to overcome these scalability issues by presenting developers with either a best solution or best set of solutions. However, such an approach does not allow developers to maintain involvement and control, resulting in a trade-off between scalability, speed, and ease of use, with innovation and ingenuity.

2.5 Operations Research

The methods discussed so far either provide quantitative evaluation techniques, or attempt to determine the best solution by applying decision criteria to previously determined alternatives. The techniques that are capable of determining alternatives on their own limit graph relationships to AND/OR decompositions. A solution to this shortcoming is the application of mathematical optimisation, a technique that builds a constraint based model that is mathematically evaluated to find the actual optimal solution. Finding an optimal solution in this manner falls in the domain of *Operations Research*, a discipline that applies advanced analytical methods to the decision making process [50, 51, 52, 53, 54].

One of the only pieces of work combining goal models with operations research has been completed by [55]. The work explores the Multi-Criteria Decision Analysis (MCDA) aspect of operations research and how it may relate to goal models, though the work still relies of the predetermination of alternatives.

Goal Programming is a branch of MCDA that has been explored by [56], building on the authors previous work [57]. This work explores a multi-objective approach to the *i* framework* through the use of a goal programming approach and integration of the IBMCplex Optimisation tool. While the work does arrive at an optimal solution, it does not support AND decompositions.

2.6 Mathematical Optimisation in Goal Orientated Requirements Engineering

The work presented in this thesis is based upon operations research, specifically, mathematical optimisation. This section is designed to provide readers with the background knowledge necessary to understand the presented work. The provided information is required for all remaining chapters, hence, the discussion has been abstracted here to prevent redundancy and provide a quick source of reference.

2.6.1 Mathematical Optimisation

Mathematical Optimisation is a broad classification of techniques that aim to select the most appropriate element from a set of options, subject to certain criteria. One of the most recognisable forms this takes is to maximise (or minimise) a goal, with respect to a series of constraints. One application of this is called linear programming or linear optimisation, a concept originally introduced by [58, 59] with roots based in, *Fourier-Motzkin Elimination (FME)* method. Since its introduction, linear optimisation has continued to be a prevalent concept as can be seen in [53, 54].

As the name suggests, linear optimisation requires that all variables involved in the model conform only to linear mathematical relationships: all variables may only be multiplied by a scalar value or constant, may not be raised to a power greater than one, and may not be used in a product calculation with another variable.

The optimisation models presented in this work are based on either linear programming or mixed integer linear programming, a variant of linear programming that limits some variables to integer values while allowing others to take non-integer form.

2.6.1.1 Terminology

Thus far, the term *optimisation model* has been used rather loosely to refer to the idea of mathematical optimisation. While this suited the discussion, it is necessary to distinguish between an optimisation model for a specific graph and the models developed in this work. *Optimisation schema* will now be used in reference to the objective function, variables, constants, and constraints, that have been developed as a conceptual idea capable of modelling any implementation of the given framework or language. The schema contains the blueprints of the equations and instructions on when an equation should be applied. For example, a schema will contain an equation for how to model an *AND* relationship. *Optimisation model* refers to the specific model that is generated based on the schema in regards to an actual graph. The model takes the blueprints of the equations and fills in the details. Section 2.6.3.2 provide an example schema and the steps necessary to generate a model based on a graph.

Distinguishing between schema and model clarifies certain aspects of this work; however, there is additional terminology that must be defined in order to facilitate any discussion of optimisation. Table 2.1 presents these terms and their definitions.

2.6.1.2 Mutual Exclusion

Many situations present in GORE models require a decision be made between mutually exclusive elements, for example: selecting the minimum or maximum element of a set. Equation (2.1) shows how mutual exclusion is typically represented mathematically by ensuring that the product of the variables is zero. However, this situation

Term	Definition
Objective Function	A linear function describing the goal of the optimisation schema, including an indication specifying if the result should be maximised or minimised.
Variables	Represent elements of the graph. Once a specific model has been generated, the optimisation software or tool manipulates the value of the variables in order to determine a result for the objective function.
Constants	A specific type of variable with a fixed value, in the schema these values are represented by variables names, in the model they are replaced with the associated values. Hence, the optimisation software will have no effect on these values.
Constraints	Equations, typically inequalities, that express relationships between variables and outline the limitations any solution must conform to. In a schema these constraints are expressed in terms of the variables and constants, in a model multiple versions of a given constraint may exist, each representing an aspect of the graph.
Solution	A solution to an optimisation model is defined by the values assigned to all non constant variables, a solution must satisfy all constraints. The *best result* is the solution that provides the highest or lowest value for the objective function, depending on if the function should be maximised or minimised.

Table 2.1: The terminology and definitions applicable to optimisation as utilised throughout the remainder of this work.

violates the rules of linear programming, as previously specified. Equation (2.2) shows an approximation of this calculation using linear relationships and the use of the *binary variables* $xFlag$ and $yFlag$; the approximation holds if x and y cannot be negative and have an upper bound, as is true in the applications of this work. The value of the constant M holds little significance as long as it is larger than the upper bound of x and y.

$$x \cdot y = 0 \tag{2.1}$$

$$M \cdot xFlag \geq x \tag{2.2a}$$

$$M \cdot yFlag \geq y \tag{2.2b}$$

$$xFlag + yFlag \leq 1 \tag{2.2c}$$

$$\text{subject to:}$$

$$xFlag, yFlag \in \{0, 1\}$$

These equations function in the same manner as all optimisation inequalities, rather than being calculated linearly they present a series of constraints all of which must hold true at any given time. Table 2.2 demonstrates how equation (2.2a) is used to determine the value of the binary variable $xFlag$ dependant on the value of variable x. For readability, the table uses integer values to demonstrate that $xFlag$ may only take a zero value *if and only if* x also has a zero value. Hence, equation (2.2c) can now ensure that only one of the flags is allowed to be non zero; if both flags have a value of one then their sum is two which would violate the constraint.

This discussion has demonstrated that given our situation it is possible to model mutual exclusion using linear inequalities. While the schema specified only three constraints, a model would need to implement these constraints for every occurrence of mutual exclusion. As the complexity of a model is determined by the number of variables and constraints it contains, this would quickly result in extremely complex models that would take significant processing to solve.

Mutual Exclusion though is hardly a new concept and this is something that *most*

x	$xFlag$	Constraint	
$x = 1$	$xFlag = 0$	$M \cdot 0 \geq 1$	false
		$0 \geq 1$	
$x = 1$	$xFlag = 1$	$M \cdot 1 \geq 1$	true
		$M \geq 1$	
$x = 0$	$xFlag = 0$	$M \cdot 1 \geq 0$	true
		$0 \geq 0$	
$x = 0$	$xFlag = 1$	$M \cdot 1 \geq 0$	true
		$M \geq 0$	

Table 2.2: Demonstration of how equation (2.2) is applied to enforce mutual exclusion.

optimisation solvers take into account by allowing for the specification of *special ordered sets (SOS)*. These sets place various restrictions on variables taking non-zero values [60]; depending on the solver there may be multiple versions of these sets with differing rules, though allowing only a single non-zero value is the most common manifestation. The advantage of special ordered sets over constraints resides in the evaluation of the models, when using constraints a branch must be explored, calculated, and then ruled out, special order sets on the other hand *prevent* evaluation of branches that it knows are invalid.

It has been demonstrated that the mutual exclusion required by the schemas presented in later chapters can be modelled not only by using linear inequalities but also through the use of special ordered sets. As such, to aid in readability and reduce the number of equations required to present a schema, mutual exclusion shall be represented in the same manner as equation (2.1). Unless otherwise specified, when implementing schemas in order to gather results, special ordered sets were utilised.

2.6.2 Theoretical Application to GORE

Chapter 2 demonstrated that quantitative methods of GORE evaluation tend to follow the same approach as their qualitative counterparts. The concomitant of this

convergent approach is that quantitative models are often subject to the same flaws suffered by their qualitative counterparts. This means, quantitative models merely propagate values to other nodes, there is no selection applied that guarantees the best possible result. Commonly this manifests itself by developers having to assign root node values — an approach subject to scalability — or assign goal node values and try to work backwards to a solution.

All of these methods rely on developers making decisions, which in large graphs can be difficult and susceptible to error, especially as the number of relationships increase. Linear Optimisation provides a unique solution to this short coming; modelling a problem as a series of constraints that must be satisfied minimises the risk of human error as they no longer need to keep track of every interdependency. In order to solve an optimisation model the solver explores every possible solution, hence, if the model is correctly formed the result is guaranteed to provide the best solution.

2.6.2.1 Conflict Resolution

A major flaw in any quantitative based evaluation technique for GORE models is the innate subjectivity of the values used. Occasionally, it may be possible to quantitatively evaluate a method and determine its exact effect, for example; access time is usually computational. More frequently such values are really quantitative representations of qualitative measures, for example; the detriment that verification techniques have on user-friendliness. This inability to accurately measure most quantitative values often creates a point of contention in software development teams, as it is unlikely every member of a team will agree with a decision. While compromises can be reached, they are, essentially, disregarding some expert opinions of the group.

The implementation of an optimisation model allows for a unique resolution to this situation through use of a *sensitivity analysis*. A sensitivity analysis is a method used in measuring the uncertainty of a *solution* by determining the degree to which any

variable can vary without altering the solution. The application of this technique to GORE models means that when a solution is calculated based on a single value, it will in fact hold for a range of values. Under this technique, as long as a developer's preferred value falls in the sensitivity range, then the solution remains valid. If a potential value exceeds the range provided by the sensitivity analysis, developers can be alerted and then intercede to handle the problem. Possible solutions to this problem include re-examining the parameter to correctly determine the value, or running the simulation again with the out of range value and comparing the results to determine the difference in solutions.

A traditional sensitivity analysis has several drawbacks when dealing with *mixed integer linear programming*. These issues will be addressed in chapter 3, specifically section 3.4 will show how these issues were overcome and the necessary results produced.

2.6.3 Practical Application to GORE

This section provides a guide to understanding the schema specifications provided in later chapters and how the schema can be used in conjunction with a graph to generate a specific model. The provided example is simplistic in nature and intended only to demonstrate the syntax used in later chapters.

2.6.3.1 Schema Presentation

There are two main parts to any of the presented schemas, the table or tables specifying the variables and constants, and the equations outlining the relationships between these variables. The equations may also be accompanied by conditions specifying when they are to be applied. In the presented schema these conditions have also been incorporated into a flowchart. There are several reasons for this abstraction; the graphical navigation makes it easier to comprehend larger models, additionally, the circumstances calling for an equation to be applied may be multi

Name	Variable	Type	Range	Description	Equations
$value_{(i)}$	Variable	Real	(0, 100)	The value of node i	equations (2.3) and (2.4)
N	Constant	Set		The set of nodes for a given graph	equation (2.4)
$C_{(i)}$	Constant	Set		The set of children nodes for node i	equation (2.3)
$weight_{(i)}$	Constant	Real	(0, 1)	The weight assigned to node i. If not specified default value is 0.	equation (2.4)

Table 2.3: Example of how variables and constants are defined in an optimisation schema.

layered, or too complex to present in a single condition. Conditions are always explained in the accompanying description of each equation.

Table 2.3 shows an example table specifying the variables that shall be used and a description of their meaning; additional data may include data types, variable bounds, and equation references. Similar to special ordered sets, a variable's range is specified separately to any constraints and prevents evaluation of invalid branches. Equation (2.3) shows that the value of a given node i can be calculated by taking the average of all its children, as represented by the set $C_{(i)}$ and using the magnitude ($|\ |$) of the set. The condition and the summation range prevents a model from attempting to take the average of an empty set, that is, a childless node's value cannot be determined by its children.

$$\text{For every node } i \text{ where } |C_{(i)}| > 0 \text{ define:}$$
$$value_{(i)} = \sum_{(j \in C_{(i)}} \frac{value_{(j)}}{|C_{(i)}|}) \tag{2.3}$$

Due to the nature of optimisation, taking the average is expressed as applying a constant fractional multiplier to each variable. Mathematically this is no different than calculating the sum and then dividing by the number of elements. Many opti-

misation solvers use a matrix to represent variables so it is not possible to represent the later method. Hence, to follow convention and assist in the implementation of these schema, they all follow the same principle of representing constraints through the addition of '$constant \cdot variable$' products. Subtraction is still represented as expected, though in implementation requires a -1 multiplier.

In order to apply an optimisation process, an objective function must also be specified. Equation (2.4) shows an objective function that will try to create the highest possible value for a graph based on the value of all nodes, as indicated by $i \in N$, with respect to their weight.

$$\max \sum_{(i \in N)} (weight_{(i)} \cdot value_{(i)}) \tag{2.4}$$

2.6.3.2 Model Generation

Now that an example schema has been created, it is possible to show how a model would be generated based on said schema. Figure 2.1 shows a basic six node graph with a single root, table 2.4 shows the associated variables and their values. Using this graph, equations (2.5) and (2.6) show the equations that would be generated based on equation (2.3). These equations were generated because of the six nodes, only node A and node C satisfy the condition of having children. The place holders i and j were substituted with the appropriate node names, the schema also used the constant value $|C_{(i)}|$. Depending on node i this constant has a different value, hence, $|C_{(A)}| = 3$ and $|C_{(C)}| = 2$ was also substituted, resulting in the two specified equations.

$$value_{(A)} = \frac{value_{(B)}}{3} + \frac{value_{(C)}}{3} + \frac{value_{(D)}}{3} \tag{2.5}$$

$$value_{(C)} = \frac{value_{(E)}}{2} + \frac{value_{(F)}}{2} \tag{2.6}$$

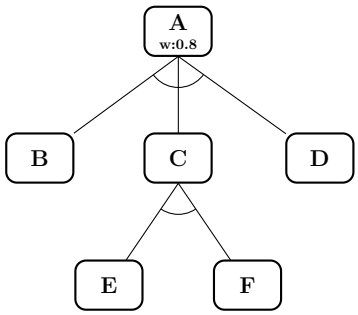


Figure 2.1: A simple six node graph with a single root node, 3 immediate children and 2 grandchildren.

Table 2.4: The variables and values generated for figure 2.1 as based on the structure defined in table 2.3

Node	Variable	Value
	N	$\{A, B, C, D, E, F\}$
A	$value_{(A)}$	TBD
	$weight_{(A)}$	0.8
	$C_{(A)}$	$\{B, C, D\}$
B	$value_{(B)}$	TBD
	$weight_{(B)}$	0
	$C_{(B)}$	$\{\}$
C	$value_{(C)}$	TBD
	$weight_{(C)}$	0.8
	$C_{(C)}$	$\{E, F\}$
D	$value_{(D)}$	TBD
	$weight_{(D)}$	0
	$C_{(D)}$	$\{\}$
E	$value_{(E)}$	TBD
	$weight_{(E)}$	0
	$C_{(E)}$	$\{\}$
F	$value_{(F)}$	TBD
	$weight_{(F)}$	0
	$C_{(F)}$	$\{\}$

At this point, four variables $value_{(B)}$, $value_{(D)}$, $value_{(E)}$, and $value_{(F)}$ have not been assigned a value, neither has their value been limited by a constraint. Hence, the optimisation solver will manipulate the value of these variables in order to achieve the objective function. If equation (2.4) specifies the schema objective function, then the model's objective function can be generated as shown step by step in equation (2.7).

$$\begin{aligned} \max \quad & (weight_{(A)} \cdot value_{(A)}) + (weight_{(B)} \cdot value_{(B)}) \\ & + (weight_{(C)} \cdot value_{(C)}) + (weight_{(D)} \cdot value_{(D)}) \\ & + (weight_{(E)} \cdot value_{(E)}) + (weight_{(F)} \cdot value_{(F)}) \end{aligned} \tag{2.7a}$$

$$\begin{aligned} \max \quad & (0.8 \cdot value_{(A)}) + (0 \cdot value_{(B)}) + (0 \cdot value_{(C)}) \\ & + (0 \cdot value_{(D)}) + (0 \cdot value_{(E)}) + (0 \cdot value_{(F)}) \end{aligned} \tag{2.7b}$$

$$\max \quad (0.8 \cdot value_{(A)}) \tag{2.7c}$$

Given this example, the optimisation solver would determine that the best solution is achieved when the four variables $value_{(B)}$, $value_{(D)}$, $value_{(E)}$, and $value_{(F)}$ are assigned a value of 100, in turn allowing $value_{(A)}$, and $value_{(C)}$ to take the value 100. In this case optimisation was not needed to determine the solution, but the models and graphs used in future chapters, and their intended application in the real world, generally involve complicated relationships and trade-offs, making manual calculation difficult at best, if not impossible.

2.6.4 Summary

This chapter provided an introduction to mathematical optimisation, its associated terminology, and some of the core concepts utilised in this work. Additionally, section 2.6.2 expanded on the shortfalls detailed in chapter 2 and elucidated how the optimisation schemas presented forthwith aim to overcome them. The remainder

of the chapter was dedicated to establishing the format and conventions followed throughout this work, and detailing how a schema can be used to generate an optimisation model based on a singular graph.

The next chapter covers the first optimisation schema to be presented in this work, based on the *Non-Functional Requirements* framework. The chapter develops the optimisation schema, applies the schema to several exemplars, expands on the aforementioned sensitivity analysis, and examines the scalability and application of the optimisation schema.

Chapter 3

Non-Functional Requirements Framework

Section 2.6 discussed the benefits of applying *Linear Programming* — specifically *Mixed Integer Linear Programming* — to *Goal Oriented Requirements Engineering* models. Additionally, the chapter established the terminology, syntax, and processes utilised by this work to present and apply an optimisation schema.

This chapter applies the discussed principles to present an optimisation schema for the *Non-Functional Requirements (NFR)* Framework; a specific implementation of the *GORE* methodology. Section 3.1 establishes the basic structure of the framework, section 3.2 presents the proposed optimisation schema, and section 3.3 propagates the optimisation results to the remaining nodes. Based on the optimisation schema, section 3.4 designs a process to extract sensitivity analysis data, and section 3.5 applies the optimisation schema and sensitivity analysis to several graphs, and interprets the results. Once the results have been achieved, section 3.6 investigates the scalability, computation time, and automation of both the optimisation schema and sensitivity analysis. Finally, section 3.7 compares these results to a published quantitative evaluation alternative, and section 3.8 details the major lessons that were learned from developing and implementing the work presented in this chapter.

The work presented in this chapter has been published in the *Oxford Computer Journal* [61], the initial concepts were also presented at the *Asia-Pacific Conference on Conceptual Modelling* [62].

3.1 The Non-Functional Requirements Framework

The NFR Framework was defined in [25], the goal of the Framework is to construct a graph highlighting the desired non-functional aspects of the system — called a softgoal — and the related implementation methods — called operationalizations. These two groups of nodes are linked using positive and negative interdependencies, hence, these graphs are called *Softgoal Interdependency Graphs (SIGs)*. Traditional evaluation methods qualitatively weigh these interdependencies and rely on developer judgement to determine which operationalizations should be implemented. This process effectively allows non-functional requirements to become a deciding factor when making design decisions.

Table 3.1 shows the graphical representation of the elements used to model Softgoal Interdependency Graphs, as they will be used in later sections of this chapter. While the original framework uses different relationships to model positive and negative interdependencies, the optimisation schema models these differences through numerical values. Therefore, to simplify the graphs and aid in readability, the multitude of relationships are replaced with a single relationship call an *impact*; an impact is defined by the two nodes it connects and is always accompanied by a weight indicating its strength and effect.

Name	Symbol	Description
Softgoal		A node used to model a non-functional requirement.
Operationalization		A node used to model a design element that will affect how the system is built or even how it operates.
AND		The parent node is achieved if *all* children nodes are achieved. This relationship can only be formed between nodes of the same type.
OR		The parent node is achieved if *any* child nodes are achieved. This relationship can only be formed between nodes of the same type.
Impact		The operationalization has an impact on the softgoal, the *weight* variable indicates the degree of the impact. This relationship is also used to model relationships between nodes of the same type where a child has some affect on the parent but does not form an AND or OR relation.

Table 3.1: Elements used in the construction of Softgoal Interdependency Graphs for the NFR Framework.

3.2 NFR Optimisation Schema

This section develops an optimisation schema for the Softgoal Interdependency Graphs constructed by the NFR Framework. The optimisation schema is based on the idea of *accepting* or *rejecting* operationalizations and was heavily influenced by quantitative approaches developed in [40, 36, 41, 63, 35, 33].

A Softgoal Interdependency Graph is composed of a multitude of nodes and relationships. Figure 3.1 outlines which of the optimisation schema equations should be implemented for each node, based on its type and its relationships.

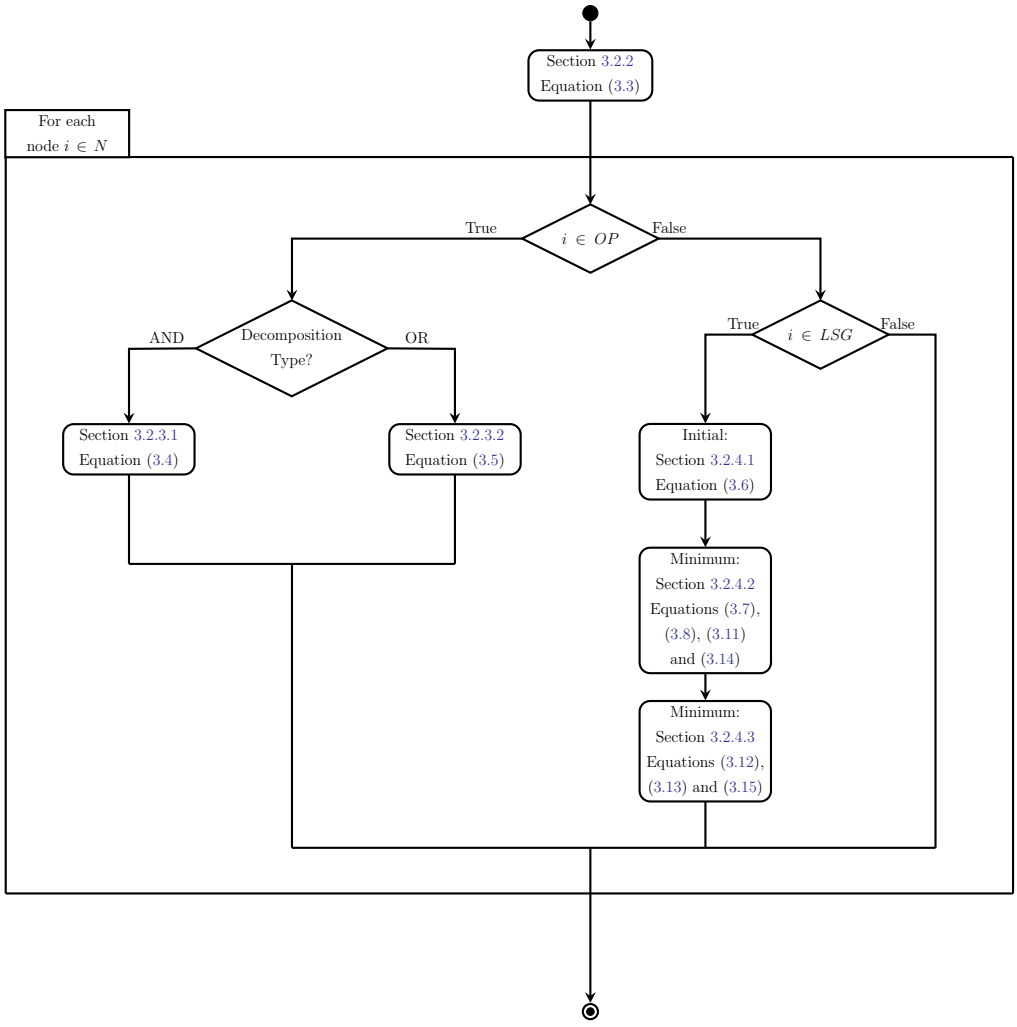


Figure 3.1: Flowchart for converting the Non-Functional Requirements Framework optimisation schema to a specific model.

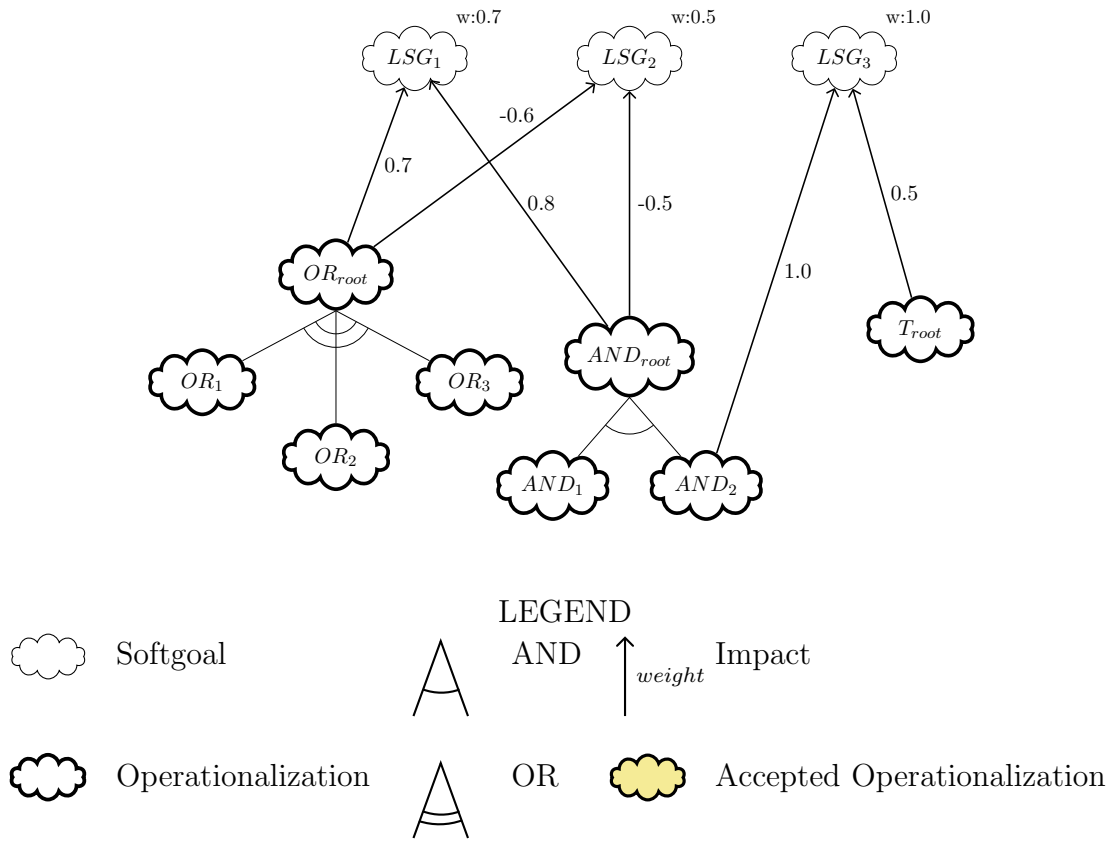


Figure 3.2: An example NFR Softgoal Interdependency Graph, used throughout section 3.2 to demonstrate the evolution of an optimisation model based on the optimisation schema

The processes modelled in figure 3.1 shall be demonstrated during the specification of the optimisation schema, based on the example Softgoal Interdependency Graph presented in figure 3.2. This graph shall be referred to as the *demonstration graph*.

3.2.1 Graph Notation of the Optimisation Schema

The structure of a Softgoal Interdependency Graph is determined by the relationships between softgoals, leaf-softgoals, and operationalizations; this structure can be modelled using the directed graph $G = (N, A)$ as defined in table 3.2; additional variables and constants are defined as needed in order to maintain readability and prevent an inundation of contextless variable names.

Variable	Explanation	Remarks
$N = SG \cup LSG \cup OP$	The sets SG, LSG, and OP respectively represent, the softgoals, leaf-softgoals, and operationalizations.	Nodes are typically denoted by i , j etc...
$A = A_1 \cup A_2 \cup A_3 \cup A_4$	A_1 — arcs within the set SG A_2 – arcs from the set LSG to set SG A_3 – arcs within the set OP A_4 – arcs from the set LSG to set OP	For an arc (i, j) between nodes i and j, node i is considered the parent, and the node j is considered the child.

Table 3.2: Graph notation used to define a Softgoal Interdependency Graph as constructed under the NFR Framework.

3.2.2 Objective Function

An objective function is used by the optimisation solver in order to determine the final values of all variables. As such, correct definition of an objective function is vital to the success of the optimisation schema. Table 3.3 defines the variables and constants required in the formulation of an objective function for the Non-functional Requirements Framework.

The objective function is based on the idea of an attainment score (denoted by A_{ideal}), calculated by equation (3.1). The ideal attainment score represents the ideal system where all leaf-softgoals have been satisficed to their full extent. The calculation is based on weight, hence, higher priority leaf-softgoals have a larger effect on the score and consequently on the objective function.

$$A_{ideal} = \sum_{\substack{i \\ i \in LSG}} LSGw_{(i)} \tag{3.1}$$

The actual attainment score (denoted by A_{actual}) is calculated by equation (3.2). This score takes into account the leaf-softgoal scores — represented by the variable $LSGs2_{(i)}$ — as they are an individual measure of how well the leaf-softgoal has been satisficed and hence, a reflection on the accepted operationalizations. Calculations

Variable	Range	Definition	Defined
$LSGw_{(i)}$	$(0.0, 1.0)$	Defined for any node $i \in LSG$. $LSGw_{(i)}$ represents the priority of the node i	Constant
A_{ideal}	$(0.0, \infty)$	The maximum possible value a given graph can achieve	Constant (3.1)
$LSGs2_{(i)}$	$(-1.0, 1.0)$	$LSGs2_{(i)}$ represents the degree to which the leaf-softgoal i has been satisficed.	(3.15)
A_{actual}	$(0.0, \infty)$	The actual value a graph achieved based on the accepted operationalization and resultant leaf-softgoal scores.	(3.2)

Table 3.3: Variables and input constants necessary to define the NFR schema objective function.

for the leaf-softgoal scores are defined in section 3.2.4

$$A_{actual} = \sum_{\substack{i \\ i \in LSG}} \left(LSGs2_{(i)} \times LSGw_{(i)} \right) \tag{3.2}$$

Given the complete graph definition and the idea of an attainment score, the optimisation schema aims to obtain the best possible attainment. The simplest method of achieving this is to create a maximisation objective function (equation (3.3)) based directly on the actual attainment score (equation (3.1)).

$$max \overbrace{\sum_{\substack{i \\ i \in LSG}} \left(LSGs2_{(i)} \times LSGw_{(i)} \right)}^{\text{From equation (3.2)}} \tag{3.3}$$

Demonstration Graph: Based on the *demonstration graph*, table 3.4 presents the constraints generated from the previous equations. Only the third constraint, that based on equation (3.3) — the objective function — is added to the optimisation model.

Variable	Range	Definition	Defined
$C_{(i)}$		The set of nodes related to the parent node i by any of the relationships defined in A. If $C_{(i)} \in SG \cup LSG$ then the relationship is either AND, OR, or $OTHER$. If $C_{(i)} \in OP$ then the relationship is either AND, or OR.	Constant
$OPa_{(i)}$	$(0, 1)$	$OPa_{(i)}$ takes the value of 1 (or 0), if the operationalization is accepted (or rejected)	(3.4), and (3.5)

Table 3.5: Notation and definitions for variables required to calculate operationalization decomposition.

Equation	**Generated Constraints**
(3.1)	$A_{ideal} = 0.7 + 0.5 + 1.0 = 2.2$
(3.2)	$A_{actual} = (LSGs2_{LSG_1} \times 0.7) + (LSGs2_{LSG_2} \times 0.5) + (LSGS2_{LSG_3} \times 1.0)$
(3.3)	$\max\big((LSGs2_{LSG_1} \times 0.7) + (LSGs2_{LSG_2} \times 0.5) + (LSGS2_{LSG_3} \times 1.0)\big)$

Table 3.4: The ideal attainment, actual attainment, and objective function generated for figure 3.2.

3.2.3 Operationalization Decomposition

When a single operationalization is too broad, it may be refined to several more detailed operationalizations. This *decomposition* may take the form of either an AND relationship or an OR relationship. While the selection of operationalizations is determined by maximum satisfaction of the objective function, the structure of this decomposition may limit the *correct* solutions available, ie: the relationships impose restrictions on which operationalizations can be accepted. These restrictions ensure that the relationships between operationalizations are not violated, in order to model these restrictions additional variables as defined in table 3.5 are required.

3.2.3.1 AND Decomposition

Equation (3.4) calculates the selection of operationalizations related by an AND decomposition; the parent node is accepted only if *all* children are accepted. Equation (3.4a) works top down to force all nodes $j \in C_{(i)}$ — for the parent node i — to be accepted if the parent should be accepted. Equation (3.4b) ensures that if all nodes $j \in C_{(i)}$ have been accepted — due to their own impacts — then node i must also be accepted.

For a node $i \in OP$ with AND relation

$$|C_{(i)}|OPa_{(i)} \leq \sum_{\substack{j \in C_{(i)} \\ (i,j) \in A_3}} OPa_{(j)} \tag{3.4a}$$

$$\sum_{\substack{j \in C_{(i)} \\ (i,j) \in A_3}} \left(OPa_{(j)}\right) - OPa_{(i)} \leq |C_{(i)}| - 1 \tag{3.4b}$$

Demonstration Graph: Based on the operationalization AND_{root} and its children from the *demonstration graph*, figure 3.1 shows that for operationalizations with the AND decomposition, equation (3.4) should be generated. Table 3.6 shows the two constraints that are generated and added to the optimisation model. As the *demonstration graph* only has one operationalization with the AND decomposition, no other constraints are generated. The range of the variables OPa would also be specified but they do not take the form of a constraint[1].

Equation	Generated Constraints
(3.4a)	$2 \times OPa_{AND_{root}} \leq OPa_{AND_1} + OPa_{AND_2}$
(3.4b)	$(OPa_{AND_1} + OPa_{AND_2}) - OPa_{AND_{root}} \leq 1$

Table 3.6: The two constraints that are generated based on figure 3.2, in order to handle AND decompositions.

[1]The complexity of an optimisation model is determined by the number of variables and the number of constraints. If variable ranges were specified as constraints, then invalid values would need to be explored and rejected. On the other hand, specifying the ranges as a hard limit prevents the solver from even exploring paths with invalid values, this also reduces the number of constraints, resulting in a more efficient optimisation model.

3.2.3.2 OR Decomposition

Equation (3.5) calculates the selection of operationalizations related by an OR decomposition; the parent node is accepted if *any* child node is accepted. Equation (3.5a) works top down to force at least one node $j \in C_{(i)}$ — for the parent node i — to be accepted if the parent should be accepted. Equation (3.5b) must be generated for *every* child node $j \in C_{(i)}$ and ensures that if node j has been accepted — due to its own impacts — then node i must also be accepted.

For a node $i \in OP$ with OR relation

$$OPa_{(i)} \leq \sum_{\substack{j \in C_{(i)} \\ (i,j) \in A_3}} OPa_{(j)} \tag{3.5a}$$

$$OPa_{(i)} \geq OPa_{(j)} \qquad \forall (i,j) \in A_3 \tag{3.5b}$$

Demonstration Graph: Based on the operationalization OR_{root} and its children from the *demonstration graph*, figure 3.1 shows that for operationalizations with the OR decomposition, equation (3.5) should be generated. Equation (3.5b) specifies that a separate constraint should be added for each child in the OR relationship hence, table 3.7 shows the *four* constraints that are generated and added to the optimisation model.

Equation	Generated Constraints
(3.5a)	$OPa_{OR_{root}} \leq OPa_{OR_1} + OPa_{OR_2} + OPa_{OR_3}$
(3.5b)	$OPa_{OR_{root}} \geq OPa_{OR_1}$
(3.5b)	$OPa_{OR_{root}} \geq OPa_{OR_2}$
(3.5b)	$OPa_{OR_{root}} \geq OPa_{OR_3}$

Table 3.7: The four constraints that are generated based on figure 3.2, in order to handle OR decompositions.

Goal	Section	Equations
Raw Leaf-Softgoal Score	Section 3.2.4.1	(3.6)
Determine if score exceeds minimum	Section 3.2.4.2	(3.7),(3.8),(3.9)
Apply minimum	Section 3.2.4.2	(3.11)
Determine if score exceeds maximum	section 3.2.4.3	(3.12),(3.13),(3.14)
Apply maximum	section 3.2.4.3	(3.15)

Table 3.8: Equation mapping to high level goals when calculation leaf-softgoal scores for the Non-Functional Requirements Framework..

3.2.4 Leaf-Softgoal Score

If a leaf-softgoal score were to be calculated programmatically it would be a straightforward task. First calculate the effect all impacts have on the leaf-softgoal and then apply a `minimum` or `maximum` function to ensure the value does not exceed the specified range. However, such functions are not available through optimisation; all values must be computed mathematically by inequalities. This means the value of a variable is determined when it is the only value for which all inequalities hold. While variables can be given bounds, they are strictly enforced and the value may *never* exceed the bound. Therefore, if a bound was used to apply a range to the value, the selected operationalizations would be limited as no leaf-softgoal could ever be "over-achieved".

Table 3.8 outlines the steps necessary to mimic the effect of a `min` or `max` function, including the sections devoted to the step and the equations used to implement the calculation. First the initial effect is calculated in an unbounded variable, then the difference between the unbounded value and the upper and lower limits is calculated and the difference used to bring the value back into the accepted range. Table 3.9 defines the additional variables needed to complete this process.

3.2.4.1 Initial Calculation

The raw leaf-softgoal score ($LSGs_{(i)}$) is calculated according to equation (3.6). The calculation is based on the value of each $impact_{(i,j)}$ and if the associated $OP_{(j)}$ of

Variable	Range	Definition	Defined
$impact_{(i,j)}$	$(-1.0, 1.0)$	Defined for an arc $(i, j) \in A_4$, this value quantifies the effect an operationalization j has on the associated leaf softgoal i.	Constant
$LSGs_{(i)}$	$(-\infty, \infty)$	$LSGs_{(i)}$ represents the initial unbounded score of leaf-softgoal i based on its associated *impact* values.	(3.6)
$n'_{(i)}$ $p'_{(i)}$	$(0.0, \infty)$	The variables $p'_{(i)}$ (the surplus) and $n'_{(i)}$ (the deficit) quantify the difference between the score $LSGs_{(i)}$ and the lower limit of -1.0.	(3.7), (3.8), (3.14)
$LSGs1_{(i)}$	$(-1.0, \infty)$	$LSGs1_{(i)}$ represents the value of a leaf-softgoal after applying the lower limit of -1.0 to the raw calculation.	(3.11)
$n_{(i)}$ $p_{(i)}$	$(0.0, \infty)$	The variables $p_{(i)}$ (the surplus) and $n_{(i)}$ (the deficit) quantify the difference between the score $LSGs1_{(i)}$ and the upper limit of 1.0.	(3.12), (3.13)

Table 3.9: Variables and input constants necessary to calculate the final score of each leaf-softgoal ($LSGs2_(i)$).

each impact is accepted, as represented by the variable $OPa_{(j)}$ — defined in the previous subsection.

$$LSGs_{(i)} = \sum_{\substack{j \in C_{(i)} \\ (i,j) \in A4}} \left(impact_{(i,j)} \times OPa_{(j)} \right) \tag{3.6}$$

Demonstration Graph: Table 3.10 shows the constraints added to the optimisation model for the *Demonstration Graph* in order to calculate the initial leaf-softgoal score. As shown in figure 3.1, the constraint modelled by equation (3.6) is generated for each leaf-softgoal in the graph.

Equation	Generated Constraints
(3.6)	$LSGs_{LSG_1} = (0.7 \times OPa_{OR_{root}}) + (0.8 \times OPa_{AND_{root}})$
(3.6)	$LSGs_{LSG_2} = (-0.6 \times OPa_{OR_{root}}) + (-0.5 \times OPa_{AND_{root}})$
(3.6)	$LSGs_{LSG_3} = (1.0 \times OPa_{AND_2}) + (0.5 \times OPa_{T_{root}})$

Table 3.10: The three constraints that are generated based on figure 3.2, in order to calculate the initial leaf-softgoal scores.

3.2.4.2 Minimum Limit

Equation (3.7) quantifies the difference between $LSGs_{(i)}$ and the lower limit of -1.0 (denied) using the variables $n'_{(i)}$ (deficit) and $p'_{(i)}$(surplus), subject to the limits in equations (3.8) and (3.9).

For every $i \in LSG$ define

$$LSGs_{(i)} + n'_{(i)} - p'_{(i)} = -1 \tag{3.7}$$

Subject to:

$$n'_{(i)} \text{ and } p'_{(i)} \geq 0 \tag{3.8}$$

$$n'_{(i)} \times p'_{(i)} = 0 \tag{3.9}$$

Equation (3.14) forces $n'_{(i)}$ and $p'_{(i)}$ to be mutually exclusive — in other words, at most one of the variables is allowed to be non-zero. This restriction is important as it prevents $n'_{(i)}$ taking a value that can be used by later equations to convert a denied score (-1.0) to a satisficed value (1.0). Section 2.6.1.2 explored how such a constraint can be implemented while maintaining linear relationships.

The constraints defined in equations (3.8) and (3.14) force the conditions presented in equation (3.10) to hold. As can be seen in these equations, the variable $n'_{(i)}$ only takes a non-zero value when the raw score ($LSGs_{(i)}$) falls below the lower limit.

$$\begin{aligned} n'_{(i)} > 0,\, p'_{(i)} = 0 &\iff LSGs_{(i)} < -1 \\ n'_{(i)} = 0,\, p'_{(i)} = 0 &\iff LSGs_{(i)} = -1 \\ n'_{(i)} = 0,\, p'_{(i)} > 0 &\iff LSGs_{(i)} > -1 \end{aligned} \tag{3.10}$$

Based on these calculations and conditions, equation (3.11) defines $LSGs1_{(i)}$ by imposing the lower limit of -1 on the raw score. As $n'_{(i)}$ takes a value equal to the shortfall, adding this deficit to the raw score increases the value and the lower limit is reached. This is the equivalent of programmatically applying `max(`$LSGs_{(i)}$`, -1.0)`.

$$\begin{aligned} &\text{For every } i \in LSG \text{ define} \\ &LSGs1_{(i)} = LSGs_{(i)} + n'_{(i)} \\ &\text{Thus we have,} \\ &LSGs1_{(i)} = -1 \iff LSGs_{(i)} \leq -1 \\ &LSGs1_{(i)} = LSGs_{(i)} \iff LSGs_{(i)} > -1 \end{aligned} \tag{3.11}$$

Demonstration Graph: Table 3.11 shows the constraints added to the optimisation model for the *Demonstration Graph*, in order to apply the minimum limit. As shown in figure 3.1, the constraints modelled by equations (3.7) and (3.11) are generated for each leaf-softgoal in the graph. Equations (3.8) and (3.9) are not modelled as constraints, rather they are specified as variable limits, and special ordered sets.

Equation	Generated Constraints
(3.7)	$LSGs_{LSG_1} + n'_{LSG_1} - p'_{LSG_1} = -1$
(3.11)	$LSGs1_{LSG_1} = LSGs_{LSG_1} + n'_{LSG_1}$
(3.7)	$LSGs_{LSG_2} + n'_{LSG_2} - p'_{LSG_2} = -1$
(3.11)	$LSGs1_{LSG_2} = LSGs_{LSG_2} + n'_{LSG_2}$
(3.7)	$LSGs_{LSG_3} + n'_{LSG_3} - p'_{LSG_3} = -1$
(3.11)	$LSGs1_{LSG_3} = LSGs_{LSG_3} + n'_{LSG_3}$

Table 3.11: The six constraints that are generated based on figure 3.2, in order to apply the minimum limit to leaf-softgoal scores.

3.2.4.3 Maximum Limit

Next, the difference between $LSGs1_{(i)}$ and the upper limit of 1.0 (satisficed) is quantified by equation (3.12) using the variables $n_{(i)}$ and $p_{(i)}$, subject to the limits presented in equations (3.13) and (3.14).

For every $i \in LSG$ define

$$LSGs1_{(i)} + n_{(i)} - p_{(i)} = 1 \tag{3.12}$$

$$n_{(i)} \text{ and } p_{(i)} \geq 0 \tag{3.13}$$

$$n_{(i)} \times p_{(i)} = 0 \tag{3.14}$$

Thus we have,

$$n_{(i)} > 0,\ p_{(i)} = 0 \iff LSGs1_{(i)} < 1$$

$$n_{(i)} = 0,\ p_{(i)} = 0 \iff LSGs1_{(i)} = 1$$

$$n_{(i)} = 0,\ p_{(i)} > 0 \iff LSGs1_{(i)} > 1$$

Equation (3.15) defines $LSGs2_{(i)}$ using $LSGs1_{(i)}$ and $p_{(i)}$. This is the true leaf-softgoal score that lies in the range -1.0 to 1.0. Programmatically this score would

be calculated by `min(`$LSGs1_{(i)}$`, 1.0)`.

$$\begin{gathered}\text{For every } i \in LSG \text{ define} \\ LSGs2_{(i)} = LSGs1_{(i)} - p_{(i)} \\ \text{Thus we have,} \\ LSGs2_{(i)} = 1 \iff LSGs1_{(i)} \geq 1 \\ LSGs2_{(i)} = LSGs1_{(i)} \iff LSGs1_{(i)} < 1\end{gathered} \tag{3.15}$$

Demonstration Graph: Table 3.11 shows the constraints added to the optimisation model for the *Demonstration Graph*, in order to apply the minimum limit. As shown in figure 3.1, the constraints modelled by equations (3.12) and (3.15) are generated for each leaf-softgoal in the graph. Again, equations (3.13) and (3.14) are specified as variable limits and special ordered sets.

Equation	Generated Constraints
(3.12)	$LSGs1_{LSG_1} + n_{LSG_1} - p_{LSG_1} = -1$
(3.15)	$LSGs2_{LSG_1} = LSGs1_{LSG_1} - p_{LSG_1}$
(3.12)	$LSGs_{LSG_2} + n_{LS4G_2} - p_{LSG_2} = -1$
(3.15)	$LSGs1_{LSG_2} = LSGs1_{LSG_2} - p_{LSG_2}$
(3.12)	$LSGs_{LSG_3} + n_{LSG_3} - p_{LSG_3} = -1$
(3.15)	$LSGs1_{LSG_3} = LSGs1_{LSG_3} - p_{LSG_3}$

Table 3.12: The six constraints that are generated based on figure 3.2, in order to apply the maximum limit to the leaf-softgoal scores.

3.2.5 Demonstration Graph:

The final optimisation model for the demonstration graph is shown in table 3.13. This optimisation model can now be provided as input to an optimisation solver, which in turn determines that the highest value of the maximisation function is achieved when, OR_{root}, AND_{root}, and AND_2 are accepted. In order to satisfy the AND decomposition constraints, AND_1, and AND_2, would both need to be

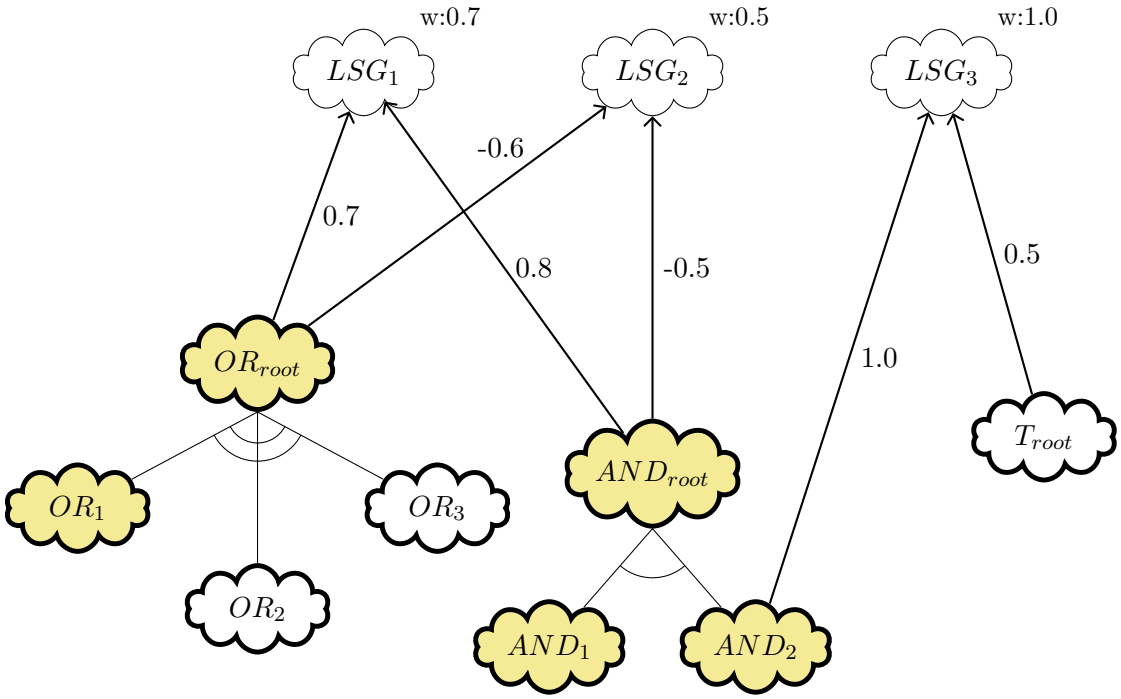


Figure 3.3: The final solution to the *Demonstration Graph* after providing an optimisation solver with the model presented in table 3.13.

accepted. The OR decomposition constraints would force only one of the OR_{root} children to be accepted [2], after the results are generated the developer could decide to change which OR_{root} child is accepted. Figure 3.3 shows the final optimisation solution to the *Demonstration Graph*.

[2]Due to how optimisation solvers are implemented, the first child to present a valid solution would be accepted, hence in this case OR_1 would be accepted as its constraints are the first ones encountered.

Equation	Generated Constraints
(3.3)	$\max\left((LSGs2_{LSG_1} \times 0.7) + (LSGs2_{LSG_2} \times 0.5) + (LSGS2_{LSG_3} \times 1.0)\right)$
	Operationalizations
(3.4a)	$2 \times OPa_{AND_{root}} \leq OPa_{AND_1} + OPa_{AND_2}$
(3.4b)	$(OPa_{AND_1} + OPa_{AND_2}) - OPa_{AND_{root}} \leq 1$
(3.5a)	$OPa_{OR_{root}} \leq OPa_{OR_1} + OPa_{OR_2} + OPa_{OR_3}$
(3.5b)	$OPa_{OR_{root}} \geq OPa_{OR_1}$
(3.5b)	$OPa_{OR_{root}} \geq OPa_{OR_2}$
(3.5b)	$OPa_{OR_{root}} \geq OPa_{OR_3}$
	Leaf-Softgoal LSG_1
(3.6)	$LSGs_{LSG_1} = (0.7 \times OPa_{OR_{root}}) + (0.8 \times OPa_{AND_{root}})$
(3.7)	$LSGs_{LSG_1} + n'_{LSG_1} - p'_{LSG_1} = -1$
(3.11)	$LSGs1_{LSG_1} = LSGs_{LSG_1} + n'_{LSG_1}$
(3.12)	$LSGs1_{LSG_1} + n_{LSG_1} - p_{LSG_1} = -1$
(3.15)	$LSGs2_{LSG_1} = LSGs1_{LSG_1} - p_{LSG_1}$
	Leaf-Softgoal LSG_2
(3.6)	$LSGs_{LSG_2} = (-0.6 \times OPa_{OR_{root}}) + (-0.5 \times OPa_{AND_{root}})$
(3.7)	$LSGs_{LSG_2} + n'_{LSG_2} - p'_{LSG_2} = -1$
(3.11)	$LSGs1_{LSG_2} = LSGs_{LSG_2} + n'_{LSG_2}$
(3.12)	$LSGs_{LSG_2} + n_{LS4G_2} - p_{LSG_2} = -1$
(3.15)	$LSGs1_{LSG_2} = LSGs1_{LSG_2} - p_{LSG_2}$
	Leaf-Softgoal LSG_3
(3.6)	$LSGs_{LSG_3} = (1.0 \times OPa_{AND_2}) + (0.5 \times OPa_{T_{root}})$
(3.7)	$LSGs_{LSG_3} + n'_{LSG_3} - p'_{LSG_3} = -1$
(3.11)	$LSGs1_{LSG_3} = LSGs_{LSG_3} + n'_{LSG_3}$
(3.12)	$LSGs_{LSG_3} + n_{LSG_3} - p_{LSG_3} = -1$
(3.15)	$LSGs1_{LSG_3} = LSGs1_{LSG_3} - p_{LSG_3}$

Table 3.13: The complete optimisation model generated for the figure 3.2.

3.2.6 Alternative Objective Function

The equations presented thus far form a complete optimisation schema for the NFR Framework. However, earlier iterations of the schema struggled to provide a linear solution to the minimum/maximum issue discussed in section 3.2.4. Until the solution was reached, a minimisation objective function was implemented and presented in [62]. The minimisation objective function can be seen in equation (3.16). While the maximisation version aims to maximise the value of each individual leaf-softgoal, this version aims to minimise the degree to which each leaf-softgoal fails to meet its maximum score.

$$\min \sum_{i \in LSG} \left(n_{(i)} \times LSGw_{(i)} \right) \tag{3.16}$$

Equation 3.2.6 presents a mathematical proof that the two objective functions are interchangeable. However, the minimisation form was necessary to prevent the optimisation solver from manipulating variables to achieve an ideal scenario.

Objective Function Proof.

$$\sum_{i \in LSG} \left(LSGs2_{(i)} \times LSGw_{(i)} \right) \tag{3.17a}$$

$$= \sum_{i \in LSG} \left(\overbrace{\left[LSGs1_{(i)} - p_{(i)} \right]}^{\text{From equation (3.15)}} \times LSGw_{(i)} \right) \tag{3.17b}$$

$$= \sum_{\substack{i \\ LSGs1_{(i)} \leq 1}} \left(\left[LSGs1_{(i)} \right] \times LSGw_{(i)} \right) + \sum_{\substack{i \\ LSGs1_{(i)} > 1}} \left(\overbrace{\left[1 - n_{(i)} \right]}^{\text{Observe } n_{(i)}=0} \times LSGw_{(i)} \right) \tag{3.17c}$$

$$= \sum_{\substack{i \\ LSGs_{(i)} \leq -1}} \left(\overbrace{\left[LSGs_{(i)} + n'_{(i)} \right]}^{\text{From equations (3.11) and (3.12), note } p_{(i)} = 0} \times LSGw_{(i)} \right) + \sum_{\substack{i \\ LSGs_{(i)} > -1 \\ LSGs_{(i)} < 1}} \left(\overbrace{\left[LSGs_{(i)} + n'_{(i)} \right]}^{\text{From equation (3.11)}} \times LSGw_{(i)} \right)$$

$$+ \sum_{\substack{i \\ LSGs_{(i)} \geq 1}} \left(\left[1 - n_{(i)} \right] \times LSGw_{(i)} \right) \tag{3.17d}$$

$$= \sum_{\substack{i \\ LSGs_{(i)} \leq -1}} \left(\overbrace{\left[1 - n_{(i)} \right]}^{\text{From equation (3.15)}} \times LSGw_{(i)} \right) + \sum_{\substack{i \\ LSGs_{(i)} > -1 \\ LSGs_{(i)} < 1}} \left(\left[1 - n_{(i)} \right] \times LSGw_{(i)} \right)$$

$$+ \sum_{\substack{i \\ LSGs_{(i)} \geq 1}} \left(\left[1 - n_{(i)} \right] \times LSGw_{(i)} \right) \tag{3.17e}$$

$$= A_{(ideal)} - \sum_{i \in LSG} \left(n_{(i)} \times LSGw_{(i)} \right) \tag{3.17f}$$

$$\therefore \min \sum_{i \in LSG} \left(n_{(i)} \times LSGw_{(i)} \right) \tag{3.17g}$$

□

3.3 Propagation of Optimisation Values

Once the optimisation has selected the operationalizations and calculated the leaf-softgoal scores, the remaining softgoal scores can be calculated. To achieve this,

Variable	Range	Definition	Defined
$contrib_{(i,j)}$	$(-1.0, 1.0)$	Defined for an arc $(i, j) \in A_2$ with the type $OTHER$, the value quantifies the contribution that a child j makes to its parent i.	Constant
$Cs_{(j)}$	$(-1.0, 1.0)$	The final value of a node j, where $j \in C_{(i)}$ given that $i \in SG$.	(3.15) *or* (3.18) *or* (3.19) *or* (3.20)

Table 3.14: Variables and input constants necessary to propagate leaf-softgoal scores to the the remaining graph nodes in the NFR Framework.

a quantitative version of the original qualitative propagation algorithm presented in [25] is introduced. This requires several of the previously defined leaf-softgoal variables to also be defined for softgoals, the names and functions of the variables are identical, except the abbreviation SG is used instead of LSG, as such, readers are directed to the previous definitions. Table 3.14 defines the additional variables required for this discussion.

As before, the calculation of softgoal scores is dependant on the type of relationship linking the parent to its children.

3.3.1 OR Decomposition

For relationships defined by an OR operation, the parent is deemed satisficed if any of the children are satisficed and denied only if all children are denied. To reflect this, and in conjunction with the qualitative propagation algorithm, the score of the parent ($SGs2_{(i)}$) is equal to the maximum score in $C_{(i)}$, this is modelled in equation (3.18).

For a node $i \in SG$ with OR operation

$$SGs2_{(i)} = max_{j \in C_{(i)}}(Cs_{(j)}) \tag{3.18}$$

3.3.2 *AND* Decomposition

For relationships defined by an AND operation, the parent is deemed satisficed only if all the children are satisficed and denied if any child is denied. To reflect this, and in conjunction with the qualitative propagation algorithm, the score of the parent ($SGs2_{(i)}$) is equal to the minimum score in $C_{(i)}$, this is modelled in equation (3.19).

For a node $i \in SG$ with AND operation

$$SGs2_{(i)} = min_{j \in C_{(i)}}(Cs_{(j)}) \tag{3.19}$$

3.3.3 *OTHER* Decomposition

The final relationship classification OTHER covers the *MAKE*, *BREAK*, *HURT*, and *HELP* contributions of the qualitative framework. Equation (3.20) outlines an additive approach that allows for full automation. Section 3.3.4 provides a complete discussion on the various potential approaches to dealing with these disjoint contributions, including the reasoning for initially adopting an additive approach, and methods of incorporating developer input. The auxiliary variables used here fall under similar constraints as those in equations (3.8), (3.13) and (3.14).

For a node $i \in SG$ with OTHER operation

$$SGs_{(i)} = \sum_{j \in C_{(i)}} Cs_{(j)} \times contrib_{(i,j)} \tag{3.20a}$$

$$SGs_{(i)} + n'_{(n)} - p'_{(i)} = -1 \tag{3.20b}$$

$$SGs1_{(i)} = SGs_{(i)} + n'_{(i)} \tag{3.20c}$$

$$SGs1_{(i)} + n_{(i)} - p_{(i)} = 1 \tag{3.20d}$$

$$SGs2_{(i)} = SGs1_{(i)} - p_{(i)} \tag{3.20e}$$

3.3.4 Softgoal Score Calculation Alternatives

The NFR Framework is based heavily on developer input, especially when it comes to label propagation. This means that while guidelines exist for calculating label values, the developer has the capacity to decide on an appropriate label as they see fit. A prime example of this is the propagation of softgoal labels by contribution type. The contribution types can range from *MAKE* to *BREAK* and combine with the child label to produce a range of effects on the parent. These effects are often contradictory in nature, and must be resolved before the parent can be assigned a label. The resolution process can be partly automated, though the developer may step in to resolve conflicts based on their expertise and domain knowledge. This results in several propagation options as outlined below.

Minimum Label: The initial approach is to collect the contribution and label combinations into a *bag*[3], the elements of the bag are then simplified and the minimal label chosen. In a quantitative optimisation schema this is calculated in the same manner as equation (3.19)

Additive: The alternative to minimum label selection is for developers to specify the strength of the contribution and hence, decide a higher label is appropriate. In the optimisation schema, this is modelled by $contrib_{(i,j)}$ which reflects the strength, and hence, type of a contribution. Negative contributions (*BREAK* and *HURT*) are modelled by negative scores, the closer the score to -1 the closer the relation is to *BREAK*. Likewise, positive contributions (*MAKE* and *HELP*) are modelled by positive scores, the closer the score to 1 the closer the relation is to *MAKE*. Equation (3.20) presents one method of allowing the strength of a contribution to impact the parent's label. In this method, a label (score) with a minimal contribution (score close to 0) has a reduced effect on the parents score.

Developer Judgement: The final method of label propagation allows the developer to examine the childrens' labels, and based on external knowledge decide on the best label for the parent. This method cannot be automated, but due to the nature of the NFR Framework, it cannot be ignored.

[3] As opposed to a *set* which does not allow duplicates.

In the interest of presenting a schema capable of full automation, the additive method was chosen and is the method utilised by the simulation in section 3.5.

3.4 Sensitivity Analysis

Section 2.6.2.1 briefly touched on the idea of applying a sensitivity analysis to a GORE model. A sensitivity analysis outlines the variance allowed by each *individual* constant before the optimal solution changes [50, 51]. Optimisation tools have the capability of automatically generating this data while calculating the optimal solution, however, obtaining sensitivity data in this manner is dependant on continuous variables. This restriction exists because maintaining integer variables exponentially increases the complexity of the problem [64].

3.4.1 Implementation

In order to overcome the afore-mentioned issue, a simulation was created that systematically tested changes in each input parameter until a change in the solution occurred. There were several steps involved in creating this sensitivity analysis, unfortunately the only option was to repeatedly calculate the optimisation model with varied values. This posed an extreme risk to the computation time of the model, fortunately the number of evaluations required was minimised by the careful selection of test values.

Traditionally, a sensitivity analysis is determined by any change in the object function value. In the context of goal graphs, this data would be meaningless, not only is the object function extremely susceptible to changes in input constants, the provided data would also be of minimal use. A far more important question that needs answering is *"When does my implementation need to change?"*, that is to say, at what point would the accepted operationalizations change. This is a question that a traditional sensitivity analysis would simply be incapable of answering.

The actual degree of accuracy required — usually the number of decimal places — may vary, for the purpose of this discussion the variable *tol* is used in place of any explicit value. Additionally, due to the nature of impacts, a value was not allowed to change from a positive($HELP$) relation to a negative($HURT$) relation, or vice versa.

3.4.1.1 Naive Approach

A simplistic approach to discovering the point at which a change occurs would be as follows:

1. Increase/decrease the given impact by *tol*.
2. Solve the new model.
3. Determine if a change occurred.
4. Repeat the process until a change occurs or the new value exceeds the allowable range.

However, this brute force approach would require a number of iterations up to the inverse order of magnitude of *tol*. For example: if $tol = 10^{-2}$, then up to 100 optimisations need to be computed *for each* impact; in the Bank system exemplar, a graph with 9 impacts, that is 900 optimisations.

3.4.1.2 Divide-and-Conquer

A more efficient approach is based on the *pivot point* strategies reminiscent of binary search methodologies [65].

1. Determine the *lower* and *upper* limits.
2. Select the midpoint.
3. Determine if a change occurred.
4. If a change occurred adjust $upper(lower)$ limit to the tested value.

If no change occurred adjust *lower*(*upper*) limit to the tested value.

5. Repeats steps 2–4 until the difference between *lower* and *upper* is less than *tol*

Algorithm 1 implements this approach in order to determine the *maximum* bound for each impact in a given graph. Algorithm 1 outlines the basic steps the simulation followed to calculate the upper bound of an impact. For readability purposes the algorithm presents the abstract process, it does not show the implementation level steps or worry about output statements and variables. It is dependant on the graph being constructed, the initial simulation executed, and the result recorded.

3.5 Results and Analysis

The previous two sections specified the optimisation schema for the NFR Framework and established a method of applying a sensitivity analysis to the results. This section will apply these processes to two prominent case studies taken from the literature, the *Bank System Exemplar* and *Keyword in Context (KWIC) Exemplar*, originally presented in [25], these systems were chosen due to their high readability and understandability. Additionally, their simplicity highlights the usefulness of undertaking a sensitivity analysis on smaller graphs.

3.5.1 Bank System Exemplar

The Bank System case study is a simplified study of the softgoals important to a bank system and some of the operationalizations used to achieve these softgoals. The exemplar includes softgoal and operationalization decomposition, and conflicting impacts. Figure 3.4 shows the initial graph, figure 3.5 shows the results of the optimisation, including the accepted operationalizations, and calculated leaf-softgoal scores. Figure 3.5 also demonstrates one method of presenting sensitivity analysis data. This is done by a parenthesised pair (min, max) that accompanies each input

Algorithm 1 Algorithm extract to calculate the maximum bound for the sensitivity analysis

```
CLASS: SensitivityAnalysis
CLASS CONSTANTS: INTERVAL=0.01, TOL=0.00001
CLASS FIELDS: graph, opActual[]

SUBMODULE: analysis()
ASSERTION: For each impact in the graph compute the lower and upper
bounds.
ALGORITHM:
for each impact i in graph do
  max := maxSensitivity(i)
  min := minSensitivity(i)
end for

SUBMODULE: maxSensitivity (impact i)
ASSERTION: For a given impact i find the upper value at which the
solution changes.
ALGORITHM:
temp := i.value, max := i.value
if (i.value > 0.0)  then
  {help and hurt have different upper bounds and exit points}
  upper = 1.0 + INTERVAL {To simulate the limit}
  inc := (1.0 - i.value)/2.0
  limit := 1.0
else
  upper := 0.0 + INTERVAL
  inc := (-1.0 * i.value)/2.0 {*-1.0 so inc is pos}
  limit := 0.0
end if

if (i.value != limit) then
  while (inc > INTERVAL) do
    max := maxSimulation(upper, inc, i)
    upper := max + inc {bound is somewhere in (max, max+inc)}
    inc := inc/2.0
    if (max = limit) then
      inc := 0.0 {upper bound did not break, stop testing}
    end if
  end while
else
  max=limit {if i.value was already a bound no need to simulate}
end if
i.value := temp
return  max

SUBMODULE: maxSimulation (limit, change, imp)
ASSERTION: Increase and simulate the impact until a change occurs or
limit is reached
ALGORITHM:
flag := TRUE
repeat
  imp.value := imp.value + change
  if (imp.value <= limit) then
    result := simulate(graph)
    for i = 0 to opActual.Count-1 do
      if (result[i].getAccept() != opActual[i].getAccept()) then
        flag := FALSE
      end if
    end for
  end if
until (flag and (imp.value < limit))
imp.value= imp.value - change {take the last correct value}
return  imp.value
```

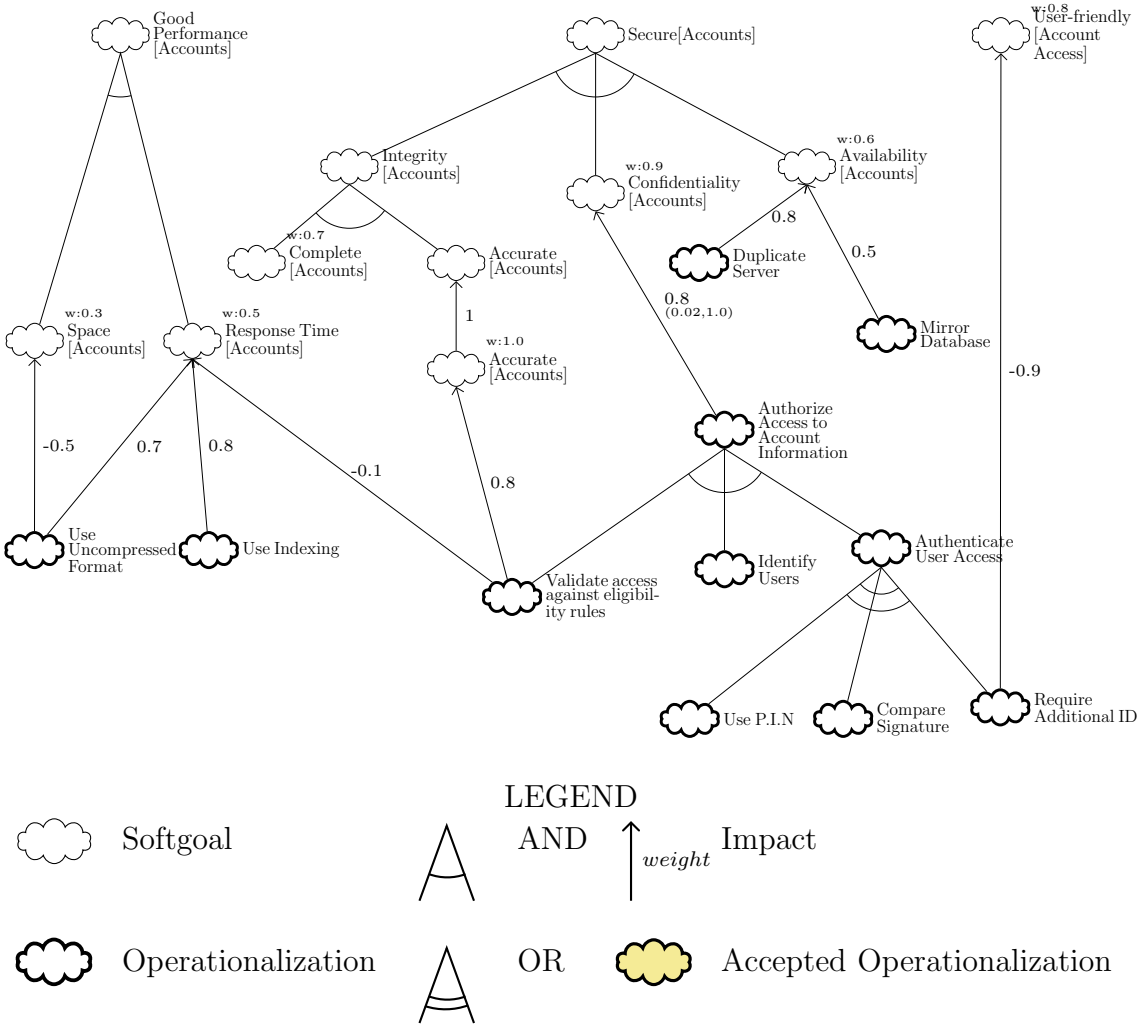


Figure 3.4: The Softgoal Interdependency Graph for the Bank System case study

impact value.

3.5.1.1 Understanding the Sensitivity Analysis

The impact from the operationalization *Use Uncompressed Format* to the leaf-softgoal *Space [Accounts]* has an input value of -0.5. According to the sensitivity analysis the impact has the bounds $(-0.5, 0.0)$, this means that the impact can take any value in this range without a change in accepted operationalizations occurring. Further data that was extracted states that if the value of the impact is lower than -0.5 then the operationalization is no longer accepted. This data can be easily inclu-

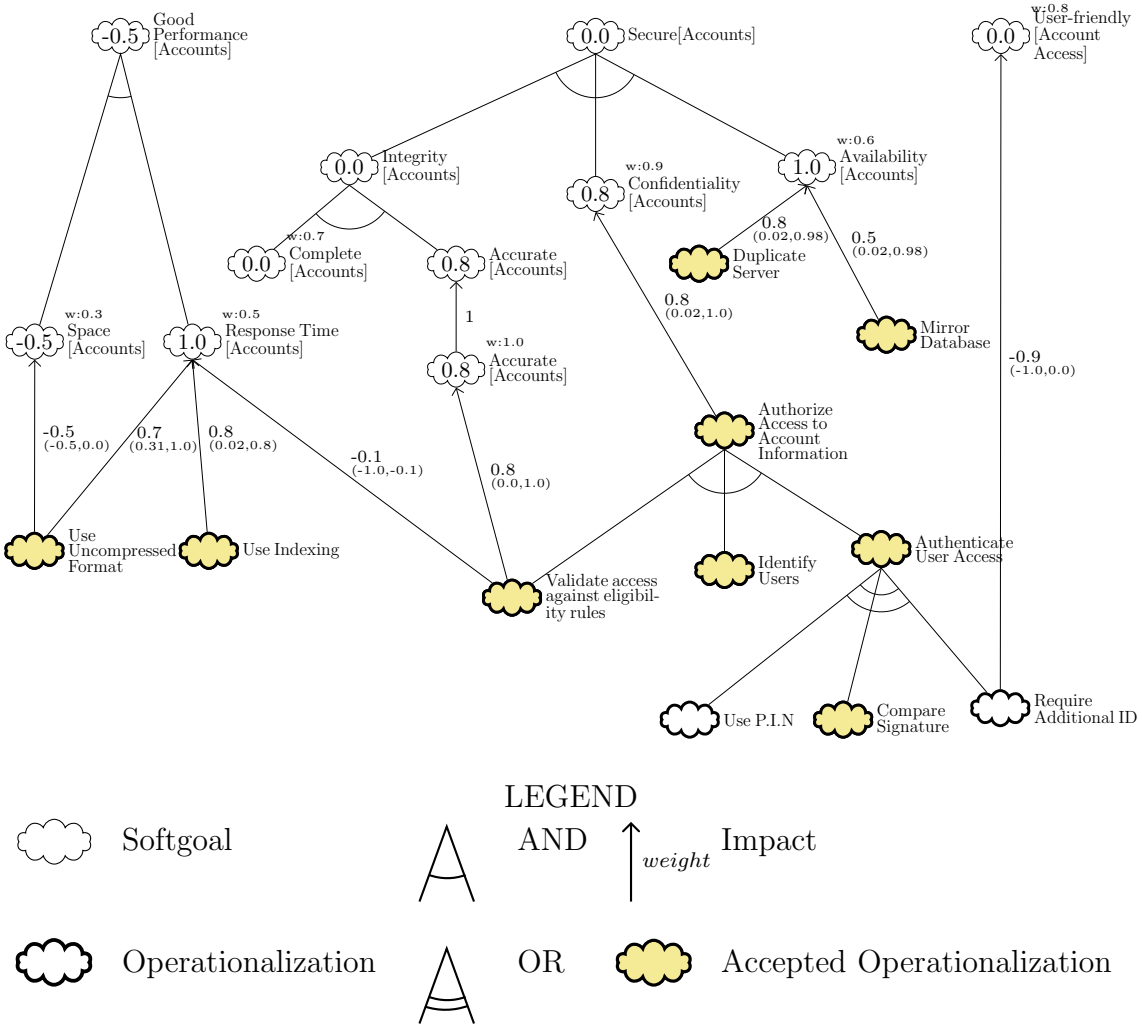


Figure 3.5: Results of the optimisation and sensitivity analysis as applied to the Bank System case study

ded in an interactive tool in order to show the different accepted operationalizations once the bounds are exceeded.

3.5.1.2 Applying the Results

The data provided by the sensitivity analysis may just seem interesting, but it actually broadens the applicability of optimisation. If there is contention between developers regarding the initial value of an impact, then the results of the sensitivity analysis can be used to settle the debate. If all values causing the problem fall in the outlined range, then the conflict can be dismissed, as regardless of the actual value chosen the *solution* would not change. In situations where a value falls outside the range, several courses of action are apparent. First, developers can analyse the solutions provided by the different inputs, from here they may come to a decision about which solution is best or try an amalgamation of both. The alternative is to decide that further discussion is needed in determining the appropriate input value, this may include the consultation of additional developers or resources.

Using the operationalization *Use Indexing* and its impact to *Response Time* as an example: two developers, Jane and Jill, have different opinions about the effect of indexing on response time. Jane thinks it will be a large help, but by itself is not enough to satisfice the NFR *Response Time*, she wants to use an impact value of 0.8. Jill however believes that indexing will only partly help, and wants to use an impact value of 0.4, they agree to run the optimisation and sensitivity analysis using an impact value of 0.8. After obtaining the results they realise that both their preferred values would result in the same operationalization selection as the sensitivity analysis provides a lower bound of 0.02. At this point Sarabi joins in the conversation and claims that indexing would indeed satisfice *Response Time*, she wants the impact to have a value of 1.0, as this value falls outside the upper bound of the sensitivity analysis (0.8) it would cause a change in the accepted operationalizations. Running the optimisation again with an impact value of 1.0 results in *Use Uncompressed Format* no longer being accepted. As this point Jane, Jill, and Sarabi would need to come to a decision over which option should be implemented.

3.5.1.3 Dual Determination of Sensitivity

The next step is to vary multiple impact values in conjunction with each other. If the contention between developers is not for a single impact, rather how changing the value of one impact would change the limits of another, then a dual analysis is the answer. Using the previous two impacts as an example, the result of this *dual analysis* can be seen in figure 3.6. This graph shows the value of the first impact (*Response Time [Accounts]*) on the x-axis and the corresponding value of *Space [Accounts]* on the y-axis.

The solid point at the base is the original impact value pair, the hollow points at the edge of the shaded area are the calculated sensitivity bounds. The vectors joining the original point to these hollow points are the test vectors. Each vector is tested in the same manner as a single impact, the shaded space between the vectors is extrapolated by connecting the test points. The accuracy of this analysis can be improved by increasing the number of vectors searched. However, increasing the accuracy does result in an increased computation time. Any impact combination that falls in the shaded region results in the same operationalizations being accepted.

Figure 3.6 shows that as long as the impact with *Space [Accounts]* does not increase, the operationalization is still accepted. Additionally, the graph shows the border where both impacts can be decreased and the operationalization still accepted. This analysis outlines that even though *Space [Accounts]* has a lower priority, it has a greater effect on the acceptance of *Use Uncompressed Format* due to the fact that *Response Time [Accounts]* has several other beneficial operationalizations.

The dual analysis data for the operationalization *Validate Access Against Eligibility Rules* and the leaf-softgoals *Response Time [Accounts]* and *Accurate [Accounts]* is shown in figure 3.7, in this case the impacts can vary by a much larger degree. This is a result of the *AND* relationship the operationalization shares with its parent; the side effect of this relationship is *Validate Access Against Eligibility Rules* indirectly being responsible for the strong, high priority impact to *Confidentiality [Accounts]*. The parental impact, in conjunction with the operationalizations own high-priority

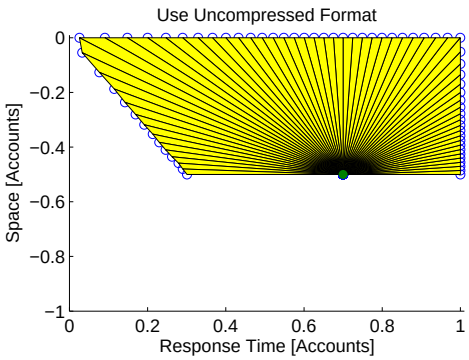


Figure 3.6: Dual comparison of impacts from *Use Uncompressed Format*

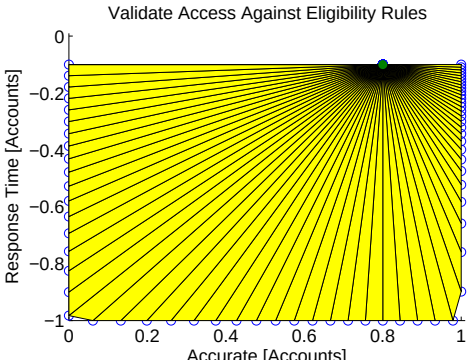


Figure 3.7: Dual comparison of impacts from *Validate Access Against Eligibility Rules*

impact, easily outweighs any negative affect the operationalization has on *Response Time [Accounts]*. In fact, additional data shows that even when the solution changes, that is, *Response Time [Accounts]* takes a value greater than -0.1, *Validate Access Against Eligibility Rules* is still accepted.

Due to the size of this case study, the initial optimisation may not have been necessary as the optimal solution can be seen by the developer. However, the addition of the sensitivity analysis provides beneficial data to developers. This data can be used in determining if the input values are satisfactory or if they need to be further refined, and has the added benefit of acting as a reliable method of conflict resolution. In both the individual and dual analysis, a change was taken to be *any* change in the accepted operationalizations, depending on the data needed a change can be defined to *only* mean a change in operationalizations in the tested impact. Additionally, the dual analysis can be expanded to consider changes in n-dimensions where n is the number of impacts investigated, including impacts from decomposed operationalizations.

3.5.2 Keyword in Context (KWIC) Exemplar

As opposed to the Bank System, Keyword in Context only models four operationalizations and has no operationalization decomposition. However, there is a signi-

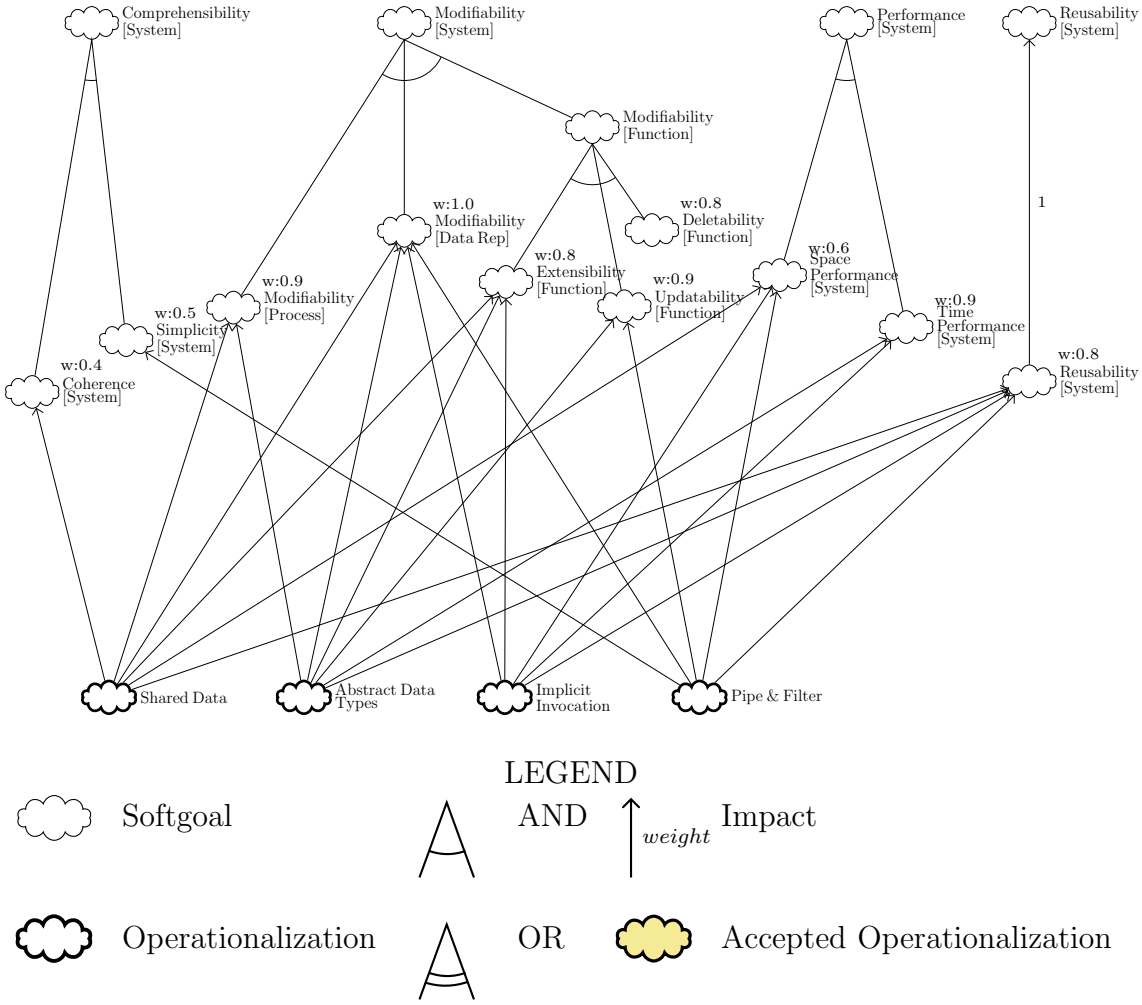


Figure 3.8: The Softgoal Interdependency Graph for the Keyword in Context System

ficantly larger number of impacts in the system, specifically those contradictory in nature. In the Bank System exemplar it was easy to tell which operationalizations should be selected, in the KWIC case study the contradicting impacts make the solution less obvious. Figure 3.8 shows the initial softgoal interdependency graph for the KWIC system.

Figure 3.9 shows the KWIC case study and the result of the initial optimisation, for readability purposes the value of each impact is not shown on the graph. The results for this case study are shown in table 3.15, here each impact is defined by the leaf-softgoal and operationalization it connects. The minimum and maximum values define the range the impact can fall in before a change occurs. As previously stated, the ranges are exclusive of any other changes. The results of the analysis on

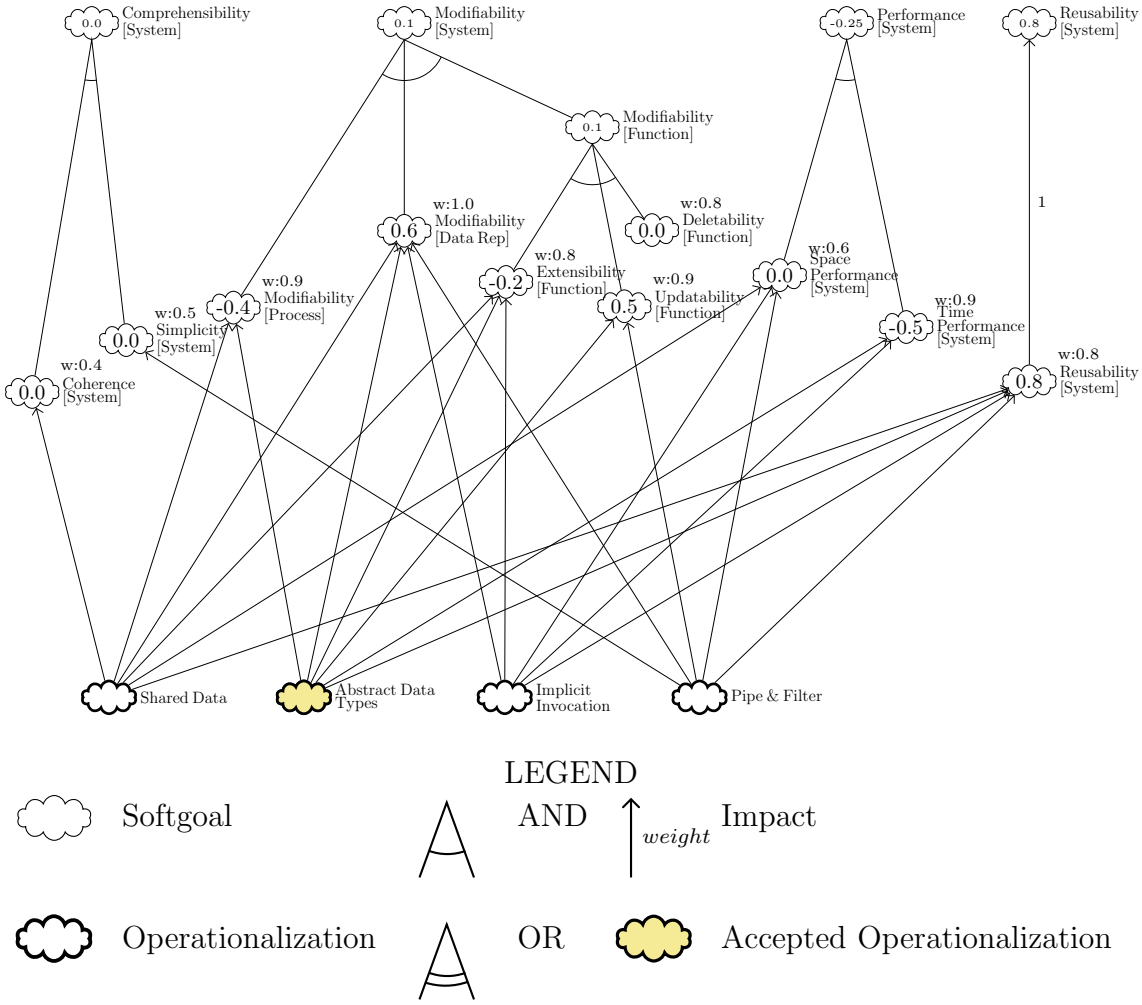


Figure 3.9: Results of the optimisation and sensitivity analysis as applied to the Keyword in Context System — NFR Framework

the KWIC system show that in most cases, the value of the impact can vary between the lower and upper limits without effecting a change in the solution. That is, as long as the relation remains either a *HURT* or *HELP* there will be no change.

3.6 Simulation

Section 3.5 applied the optimisation schema and sensitivity analysis to two exemplars and provided an interpretation of the results. It did not, however, discuss the feasibility of the process in regards to scalability and computation time, the two

Leaf-Softgoal	Operationalization	Min	Actual	Max
Coherence[System]	Shared Data	0.0	0.5	1.0
Simplicity[System]	Pipe Filter	0.0	1.0	1.0
Updatability[Function]	Abstract Data Types	0.0	0.5	1.0
Updatability[Function]	Pipe Filter	0.0	0.7	1.0
Extensibility[Function]	Abstract Data Types	-0.8	-0.2	0.0
Extensibility[Function]	Shared Data	0.0	0.4	1.0
Extensibility[Function]	Implicit Invocation	0.0	1.0	1.0
Modifiability[DataRep]	Abstract Data Types	0.112	0.6	1.0
Modifiability[DataRep]	Pipe Filter	-1.0	-1.0	-0.515
Modifiability[DataRep]	Shared Data	-1.0	-0.7	0.0
Modifiability[DataRep]	Implicit Invocation	-1.0	-0.6	-0.225
Modifiability[Process]	Abstract Data Types	-0.943	-0.4	0.0
Modifiability[Process]	Shared Data	-1.0	-1.0	0.0
Time Performance[System]	Abstract Data Types	-0.921	-0.5	0.0
Time Performance[System]	Implicit Invocation	-1.0	-1.0	-0.468
Space Performance[System]	Pipe Filter	-1.0	-1.0	-0.187
Space Performance[System]	Shared Data	0.0	1.0	1.0
Space Performance[System]	Implicit Invocation	-1.0	-0.5	0.0
Reuseability[System]	Abstract Data Types	0.2	0.8	1.0
Reuseability[System]	Pipe Filter	0.0	0.5	1.0
Reuseability[System]	Shared Data	-1	-0.8	0.0
Reuseability[System]	Implicit Invocation	0.0	0.6	1.0

Table 3.15: Results of the sensitivity analysis for the KWIC system

Test Case	Number of LSG	Number of OP	Number of Impacts	Number of AND (OP)	Number of OR (OP)
Test Case 1	25	42	468	6	7
Test Case 2	13	14	147	2	3
Test Case 3	25	32	379	6	7
Test Case 4	25	39	441	8	7
Test Case 5	13	14	75	2	3

Table 3.16: Size specifications of the simulation test graphs.

major determinants of feasibility. Additionally, the final portion of this section is a direct discussion on the ability of the optimisation schema to be automated, the discussion is directly related to the simulation created and utilised in section 3.5.

3.6.1 Simulation Test Graphs

A series of five test graphs were created through a combination of intentionally modelling key combinations and random generation. The final versions of the graphs also underwent editing to maximise the combination of input constants. Table 3.16 illustrates the final size of each of these test graphs, figure 3.10 depicts the smallest of the test cases — Test Case 5.

To ensure rules relating to operationalization decomposition were correctly handled, each of the test cases were sketched by hand, the results of the optimisation applied, and then examined. This process used the additive method of score propagation, however, it does illustrate a point in which developer intervention may be appropriate. The highlighted softgoals show how a single denied score can cause all the ancestors of an *AND* decomposition to be denied. Given that the remaining leaf-softgoal scores in the decomposition tree have positive scores, the developer could step in at this point and decide on a more appropriate score.

As test Case 5 illustrated, the test cases are complex and either comparable or larger than real world situations. While lacking real world resemblance, these graphs do establish that the optimisation schema is capable of dealing with any graph

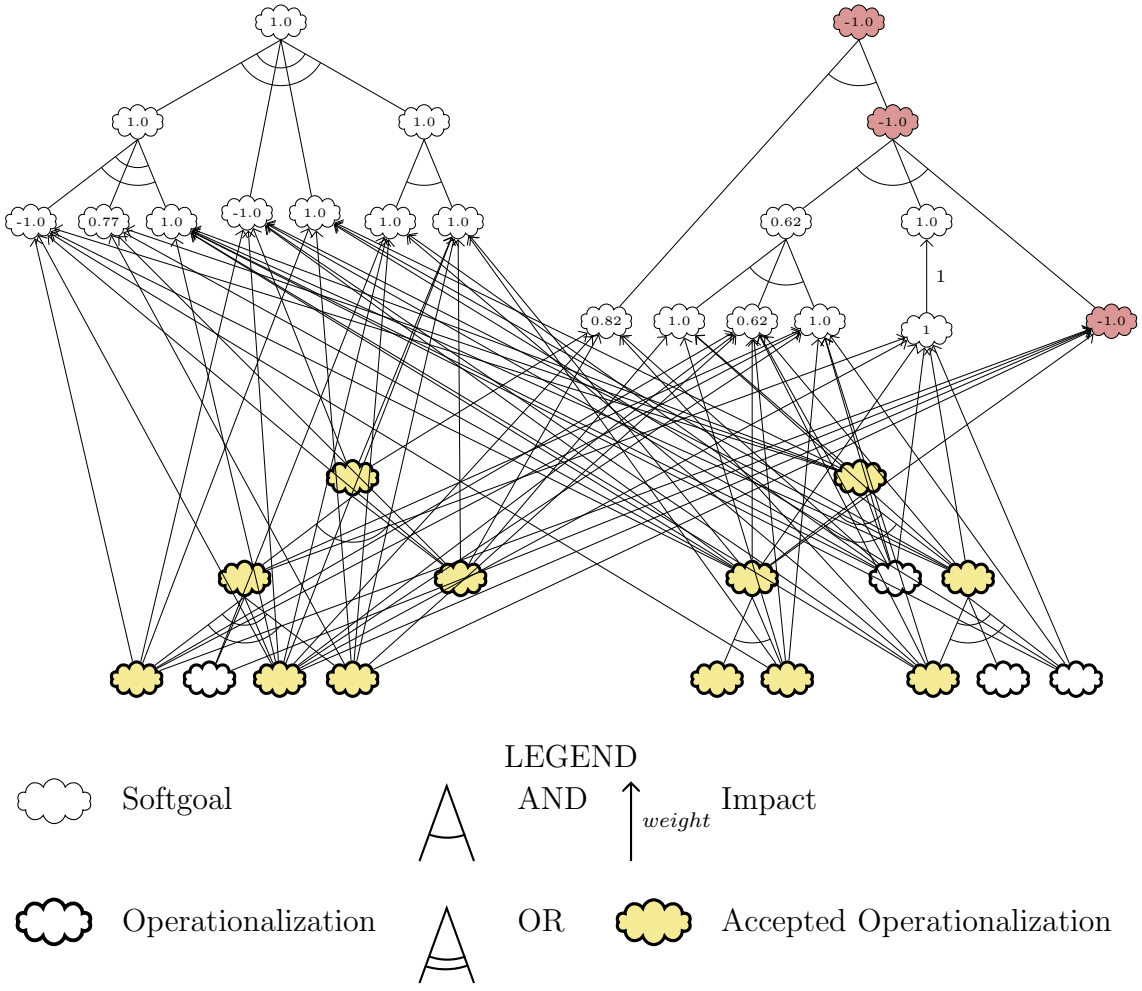


Figure 3.10: Softgoal Interdependency Graph after applying the optimisation schema to Test Case 5

Graph	No. LSG	No. OP	No. Impacts	Optimisation Time	Sensitivity Time
Bank System	7	11	8	<1s	<1s
KWIC	10	4	22	<1s	<2s
Test Case 1	25	42	468	2m 5s	>12h
Test Case 2	13	14	147	<1s	7m 44s
Test Case 3	25	32	379	<6s	7h 51m 34s
Test Case 4	25	39	441	<7s	8h 17m 38s
Test Case 5	13	14	75	<1s	46s

Table 3.17: Computation times of various test graphs. (Based on a 64-bit 3.30GHz Windows 7 desktop with 8GB RAM)

conforming to the NFR Framework.

3.6.2 Computational Time

The complexity of the test graphs proved that the optimisation schema was able to handle graphs far larger than any that would feasibly exist in the real world. However, this does not guarantee the results can be produced within a reasonable time. Table 3.17 outlines the computation time for all 6 graphs — determined on a 64-bit 3.30GHz Windows 7 desktop with 8GB RAM — compared to the specified sizes. This shows that the optimisation results can be determined quickly regardless of graph size. Computation time only become excessive when calculating the sensitivity analysis, at this point only one of the test cases failed to produce results within 12 hours. The graph in question — Test Case 1 – was the largest of the 5 test graphs, comprising 42 operationalizations and 468 impacts. Of the remaining test cases, two completed the sensitivity analysis in under 10 minutes, the final two test cases involving 379 and 441 impacts, respectively, were the only cases to exceed 10 minutes. The two exemplars suffered no time increase when calculating sensitivity than calculating optimisation.

3.6.2.1 Computation Time Options

In situations where sensitivity analysis computation times become untenable, there are still several options available.

As-needed Calculation: The sensitivity analysis does not need to be carried out every time the optimisation is completed, only when developers require access to the data it provides.

Individual Calculation: The results of a sensitivity analysis are independent and can be calculated as such. That is, when an individual impact comes under scrutiny, developers can request the bounds be calculated, *without* having to recalculate the entire graph.

Time-out: The least savoury solution of handling large graphs is to add a time-out feature to the optimisation; if the optimal solution has not been found after a specified time period, then the best solution so far is chosen. This is a feature offered by a range of optimisation solvers.

3.6.3 Notes on Implementation

The simulation was constructed using a combination of C# and LPSolve [64]. LPSolve is a free *Mixed Integer Liner Programming* solver licensed under the *GNU lesser General Public License*, meaning source code, examples, and manuals are readily available. LPSolve is capable of solving models comprised of a combination of pure linear, mixed integer, binary, semi-continuous, and special ordered sets (SOS). LPSolve has the additional benefit of interfacing with several programming languages, and the ability to read and process models from a range of alternative solvers.

The C# portion of the implementation reads a text based graph definition and then generates the appropriate constraints based on each node. Once complete, LPSolve is used to calculate the result of the optimisation model. As this optimal solution is presented as a matrix — see section 3.6.3.1 — the simulation generated several

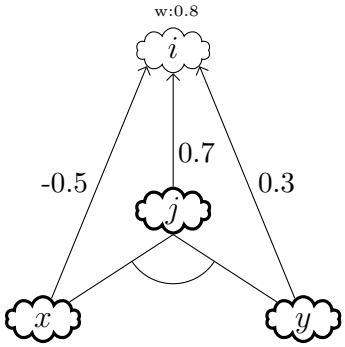


Figure 3.11: Example NFR SIG used to demonstrate a matrix optimisation model.

output files, including the final node values, $OPa_{(i)}$, and $LSGs2_{(i)}$; the softgoal values $SGs2_{(i)}$ were also calculated and output at this point. Additionally, auxiliary variables such as $LSGs_{(i)}$, $LSGs1_{(i)}$, $n'_{(i)}$, $p'_{(i)}$, $n_{(i)}$, and $p_{(i)}$ were also output, the availability of these auxiliary variables was extremely useful in isolating and correcting faults in the simulation, and in the initial versions of the optimisation schema itself.

3.6.3.1 Matrix Solutions

LPSolve — along with the majority of optimisation solvers — is a matrix solver where each variable is a column and each constraint a row; as such, the right-hand side can only take constant values. Hence, the constraints need to be restructured to comply with this format. Table 3.18 demonstrates this rearrangement using a leaf-softgoal i and operationalization j, x and y, where x and y are children of j, as shown in figure 3.11. The table shows both the AND and OR equations, however, only one set is used for a given relationship. The value in each column represents the constant that the variable (column) is multiplied by. Table 3.19 shows the generated matrix model based on the example graph, the highlighted cells show values that needed to be modified for the specific graph; the majority of constants remained unchanged, the only decision was *which* equations to implement.

Constraint	$LSGs2_{(i)}$	$LSGs1_{(i)}$	$LSGs_{(i)}$	$n'_{(i)}$	$p'_{(i)}$	$n_{(i)}$	$p_{(i)}$	$OPa_{(j)}$	$OPa_{(x)}$	$OPa_{(y)}$	Equality	RHS
Equation (3.6)			1					$-1 \cdot impact_{(i,j)}$	$-1 \cdot impact_{(i,x)}$	$-1 \cdot impact_{(i,y)}$	=	0
Equation (3.7)			1	1	-1						=	-1
Equation (3.11)		1	-1	-1							=	0
Equation (3.12)		1				1	-1				=	1
Equation (3.15)	1	-1					1				=	0
Equation (3.4a)								$\lvert C_{(i)} \rvert$	-1	-1	$\leq$	0
Equation (3.4b)								-1	1	1	$\leq$	$\lvert C_{(i)} \rvert - 1$
Equation (3.5a)								1	-1	-1	$\leq$	0
Equation (3.5b)								1	-1		$\geq$	0
Equation (3.5b)								1		-1	$\geq$	0

Table 3.18: How the NFR optimisation constraints are formatted in matrix form

Constraint	$LSGs2_{(i)}$	$LSGs1_{(i)}$	$LSGs_{(i)}$	$n'_{(i)}$	$p'_{(i)}$	$n_{(i)}$	$p_{(i)}$	$OPa_{(j)}$	$OPa_{(x)}$	$OPa_{(y)}$	Equality	RHS
Equation (3.6)			1					-0.7	0.5	-0.3	=	0
Equation (3.7)			1	1	-1						=	-1
Equation (3.11)		1	-1	-1							=	0
Equation (3.12)		1				1	-1				=	1
Equation (3.15)	1	-1					1				=	0
Equation (3.4a)								2	-1	-1	≤	0
Equation (3.4b)								-1	1	1	≤	1

Table 3.19: Matrix optimisation model of figure 3.11.

3.7 Quantitative Result Comparison

Thus far, the NFR optimisation schema has been demonstrated on several test cases, and the feasibility of its implementation ascertained. The discussion has not, however, determined if the optimisation schema presents a preferable evaluation to its quantitative counterparts.

The goal of the optimisation schema is to *selectively* accept operationalizations in order to achieve the best attainment score for a system. To determine if this goal was met, the five test cases were evaluated using a quantitative propagation method [33]. The results were then compared to determine if the optimisation schema succeeded in minimising the number of accepted operationalizations while *improving* the attainment score. Ideally this would also result in fewer leaf-softgoals receiving a negative score or being denied.

3.7.1 Operationalization Acceptance

The first goal is to selectively accept operationalizations based on their effect on the attainment score, but not to select more operationalization than necessary. While the initial purpose of selectively accepting operationalizations is to improve the leaf-softgoal scores, there is the additional benefit that limiting the number of accepted operationalizations may decrease the implementation effort or cost of a system. In order to compare the selected operationalizations, table 3.20 outlines the number of operationalizations present in each test case and the number of accepted operationalizations from both the Extended NFR Framework and the Optimisation model. In most cases the optimisation model significantly reduces the number of operationalizations accepted, however by itself this comparison is vague and does not necessarily demonstrate an improvement. Table 3.21 clearly shows that the significant reduction in accepted operationalization has resulted in a significant improvement in the attainment scores. The selective operationalization acceptance of the optimisation schema improved the attainment scores by at least 10% in all test cases.

Test Case	$\lvert OP \rvert$	ENFR $\lvert OPa_{(i)} = 1 \rvert$	Optimisation $\lvert OPa_{(i)} = 1 \rvert$
TestCase1	42	33	19
TestCase2	16	9	6
TestCase3	37	24	10
TestCase4	44	30	18
TestCase5	16	14	12

Table 3.20: Accepted operationalizations for the Extended NFR Framework and optimisation model

Test Case	A_{ideal}	ENFR A_{actual}	Optimisation A_{actual}
TestCase1	12.9	3.697	9.358
TestCase2	14.7	3.776	5.278
TestCase3	11	7.449	8.569
TestCase4	14.7	7.542	12.947
TestCase5	7.1	3.122	4.555

Table 3.21: Attainment scores for the Extended NFR Framework and optimisation model

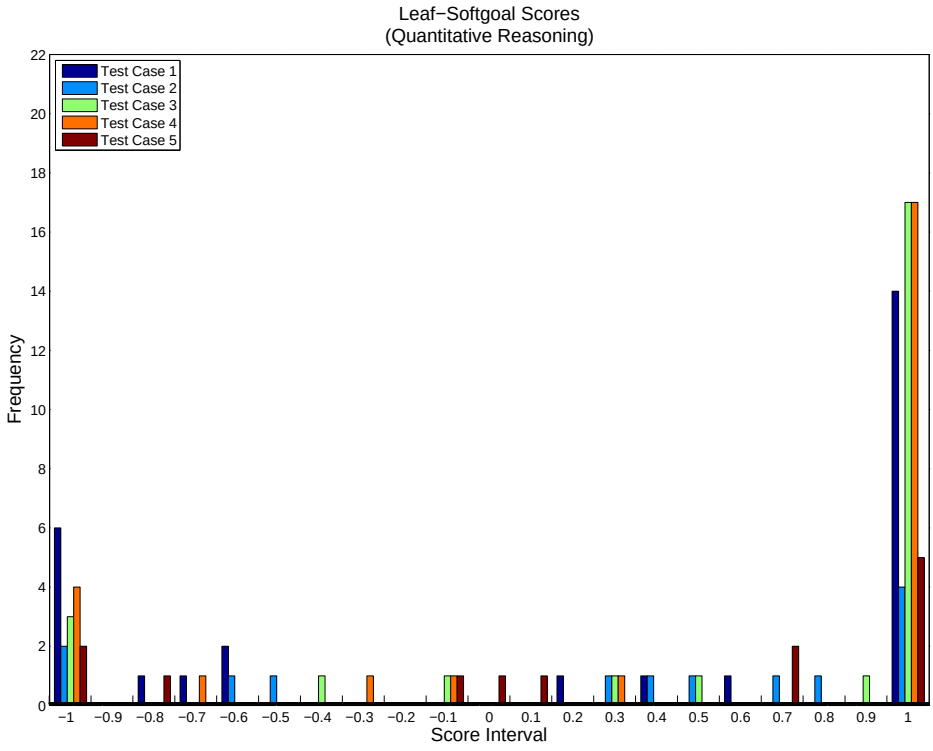


Figure 3.12: Leaf-Softgoal score distributions for the Extended NFR Framework across five test cases.

3.7.2 Denied Leaf-Softgoals

The second goal is to reduce the number of denied leaf-softgoal scores, figures 3.12 and 3.13 show the distribution of leaf-softgoal scores across 0.1 intervals for the Extended NFR Framework [33], and optimisation model respectively. A comparison of these two graphs shows that while 70% of the denied leaf-softgoals have improved their score, the number of denied scores in both cases is fairly small compared to the number of scores present. However, there is a significant shift of all scores towards the optimal score of 1, that is, the only negative scores remaining are those that have been denied.

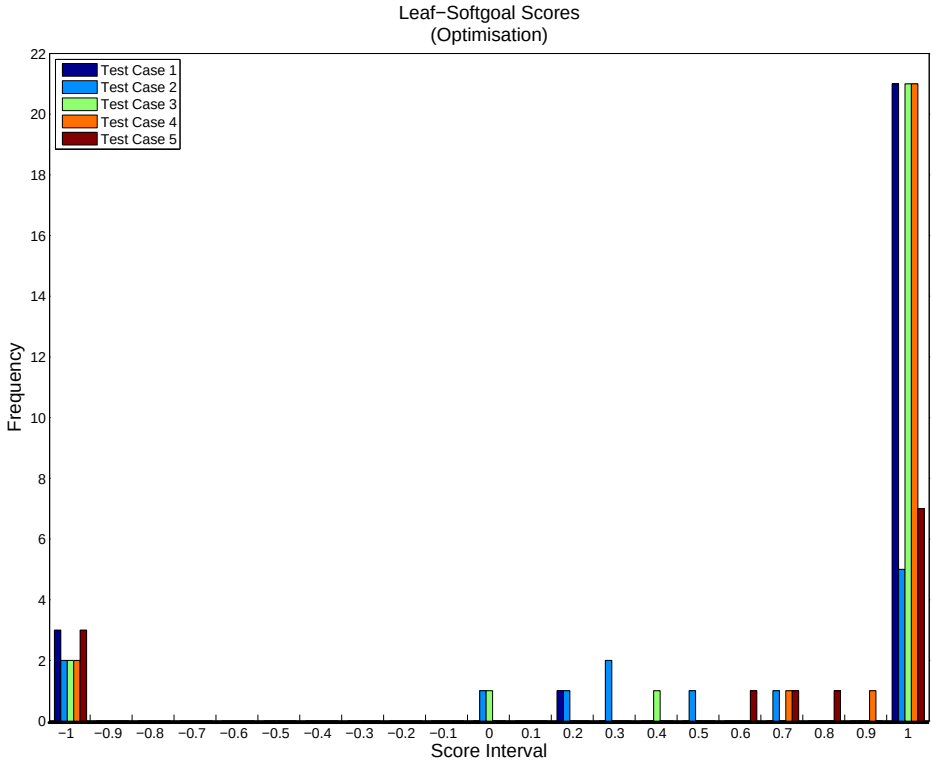


Figure 3.13: Leaf-Softgoal score distributions for the optimisation model across five test cases

3.8 Lessons Learned

Developing a linear programming optimisation schema resulted in several lessons being learned. From a broad perspective, considerable effort was put into learning linear programming, both the concept and the use of several simulators. However, there were two main points that presented significant hurdles in developing the schema. The section discusses these issues in detail.

3.8.1 Applying Limits in Linear Programming

The largest issue faced was in correctly applying limits in linear programming. From a computing perspective, this is achieved through the use of a basic *min* or *max* function call often utilising a control structure. However, linear programming does not allow for control structures to be built into the constraints. Initial attempts at constructing constraints resulted in solvers being able to assign auxiliary variables' values that result in the final leaf-softgoal score always taking the value 1.0. This manipulation was achieved by concurrently assigning both $n_{(i)}$ and $p_{(i)}$ non-zero values. The first step in overcoming this issue was to develop the minimisation objective function. This objective function changed the goal of the optimisation and forced it to always choose the smallest possible value for $n_{(1)}$. While the results appeared correct, at best the schema could claim to *approximate* a solution.

Through this process it had been discovered that the problem was the mutual exclusion of the aforementioned variables. The solution would appear straight forward in adding a constraint where the product of the variables must be zero. However, adding a constraint of this nature would result in a polynomial solution, an undesirable circumstance. Investigation into LPSolve, the solver being used to create the simulation, revealed *special ordered sets* — a function that allowed only one variable in a set to be non-zero. While the implementation of these sets is common in linear solvers they are not standardised, that is, not all solvers will implement all types of sets. Eventually, a series of constraints was devised that allowed the mutual exclusion to be modelled in a non-polynomial manner. As the solution re-

quired additional variables once it was verified, the majority of the simulations were completed using the special ordered sets as the solver was optimised for these sets and hence, computation time was significantly reduced.

3.8.2 Sensitivity Analysis with Non-Continuous Variables

After finalising the schema the remaining issue was the sensitivity analysis, as mentioned previously linear solvers provide sensitivity data that indicate the degree to which a variable can change before the solution changes. Section 3.4 discusses the implemented solution in detail and the specifics of how to create a manual sensitivity analysis. Developing the algorithm required an alternate method of determining the limits of each variable. Traditionally, this is done by recording the values as the simulation occurs, a solution that was infeasible unless a new solver was to be written. Instead values had to be tested and refined until a limit was discovered. How to do this while minimising the number of computations was the biggest issue in developing this part of the schema.

3.9 Summary

This chapter developed an optimisation schema and propagation method for the NFR Framework, additionally, a novel approach to applying a sensitivity analysis to a previously unsuitable situation was developed. Both optimisation schema and sensitivity analysis were then applied to several exemplars, the analysis of the results determined that while the optimisation process is significantly more useful in systems involving multiple conflicts, the sensitivity analysis provides useful insights in all cases; the sensitivity analysis was also expanded to simultaneously evaluate changes in multiple impacts.

The scalability and computation time of both the optimisation schema and sensitivity analysis were evaluated using a series of exceedingly large test cases, proving

that in any realistic graph both methods completed evaluation in a timely manner. Finally, the results of the optimisation schema were compared to their quantitative counterpart, and were found to produce results with a higher attainment value using less operationalizations.

Overall, the optimisation schema proved effective in evaluating graphs with a large number of contradictory relationships — a situation where human evaluation often proves infeasible. The schema prevents leaf-softgoals from being over-achieved to the detriment of their counterparts and provides a *minimal* accepted operationalization set. The inclusion of a sensitivity analysis allowed the optimisation schema to provide relevant data even on small predictable graphs.

The NFR Framework provides a goal-oriented requirements engineering method of addressing non-functional requirements. However, the rigidity of the framework severely limits the situations developers can model, the *Goal-Orientated Requirements Language (GRL)* on the other hand presents developers with elements they can combine as required, allowing the modelling of a vast range of situations. The next chapter develops an optimisation schema for GRL that maintains the flexibility of the language and provides several alternative evaluation strategies.

Chapter 4

Goal Requirements Language

The previous chapter developed and verified an optimisation schema for the Non-Functional Requirements *Framework*, a specific implementation of the GORE methodology. Chapter 2 outlined that an alternative approach to GORE was the provision of generic goal model techniques and evaluations, such as the *goal-orientated requirements language*. This chapter develops an optimisation schema for this flexible, polymorphic, and adaptable modelling tool.

Section 4.1 provides a brief introduction to GRL, section 4.2 then develops a comprehensive and flexible optimisation schema. To demonstrate the constraints used in the optimisation schema, section 4.3 shows a step-by-step analysis of their behaviour to prove they function as expected. Section 4.4 applies the optimisation schema to several test cases, using a variety of evaluation options and comparing the results, additionally, the computation time for the various evaluations is also discussed. Finally, sections 4.5 and 4.6 respectively outline the restrictions of the developed optimisation schema, and the defining lessons that were learned in its development.

4.1 Language Basics

The Goal-Oriented Requirements Language, provides a range of nodes and relationships that can be used to construct a goal model, the relationships in these graphs illustrate the effects the nodes have on each other. Table 4.1 defines the various nodes available in a GRL graph, and their associated symbols as utilised throughout this chapter. Likewise, table 4.2 defines the relationships that can be used to link

Name	Symbol	Description
Goal		Represents a concrete goal whose achievement can be easily measured. The top right hand corner shows the value of the node. The bottom right hand corner shows the weight of the node
Softgoal		Softgoals are similar to goals, but their attainment cannot be explicitly measured. They represent an idea, the most common use of softgoals is representing non-functional requirements.
Task		A task represents methods of attaining goals and softgoals.
Resource		A resource is a physical or information based item. The existence of absence or a resource may have various effects on other elements.

Table 4.1: Nodes available for use in a GRL graph.

these nodes together.

GRL not only allows the use of nodes and relationships, it also allows for the modelling of actors, an actor may represent a stakeholder or even another system. These actors can encompass any number of nodes, as such, relationships between nodes may also span between actors. Actors are a conceptual representation of ownership, as such, it is possible to calculate a value based on that of associated nodes. However, such a value provides little aid in selecting tasks and resources aimed at achieving the highest value for softgoals and goals.

4.2 Optimisation Schema

This section presents an optimisation schema for GRL, inspired by traditional qualitative evaluation techniques as illustrated in chapter 2. In order to maintain the polymorphic nature of the language, the resultant schema is more complex than the previous NFR schema, as such, careful note should be taken of the conditions applied to each constraint presented in this section. Figure 4.1 shows how the versatile

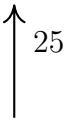
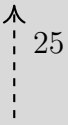
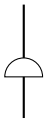
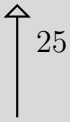
Name	Symbol	Description
Decomposition (AND)		The parent is satisfied/satisficed only when *all* children are satisfied.
Decomposition (OR)		The parent is satisfied/satisficed if *any* child is satisfied.
Contribution		The child has some effect (as indicated by the value) on the parent (positive or negative).
Correlation		Similar to contribution, but it a side effect rather than direct intention.
Dependency		As the name implies, shows how one element can be achieved by the satisfaction of another.
Means-End		Shows interdependencies between elements, when the parent element depends on the child the parents satisfaction may not exceed the child. Typically the elements belong to different actors.

Table 4.2: Relationships available for use in a GRL graph.

nature of GRL creates an optimisation schema with significantly more options than chapter 3. These options are assigned to the optimisation model, not individual nodes, they do however effect which equations from the schema will be used.

The first option ($Initialisation?$) — highlighted in blue — allows users the option to use initialisation values, any *leaf node* may be given an initialisation value that, in effect, overrides the optimisation solver. If all leaf nodes are assigned an initialisation value, then the optimisation model simply becomes a propagation algorithm, using the constraints as the propagation method.

The second option ($CappedValues?$) — highlighted in yellow — presents a difference in evaluation techniques. Many qualitative evaluation techniques subscribe to the idea that if you deny something with a negative effect, then your action was beneficial to the goal. When capped values are *not* used, the optimisation schema also subscribes to this belief. The alternative is to *use* capped values, in which case a node may only affect another if some evidence of its implementation exists, that is, it has a positive value.

The final option ($Tolerance?$) — highlighted in green — prevents numerous contribution or correlation relationships from having a collective impact on a node that results in its satisfaction or denial. By default, this option is "disabled", that is, the tolerance value is set to 100. When a tolerance value is set to say 90, then a node may only take a value higher than 90 if the impact of a contribution/correlation is greater than 90, or it gains the additional value from another type of relationship.

Demonstration Graph As in chapter 3, the presented optimisation model is accompanied by a small *demonstration graph*, designed to illustrate the process presented in figure 4.1. An indication is given for each constraint, to signify which evaluation option it is generated for[1]. At the end of the section, the complete optimisation models based on the various evaluation options shall be compared.

[1] A notation of *All*, means the constraint relates to the base schema and is generated regardless of evaluation options.

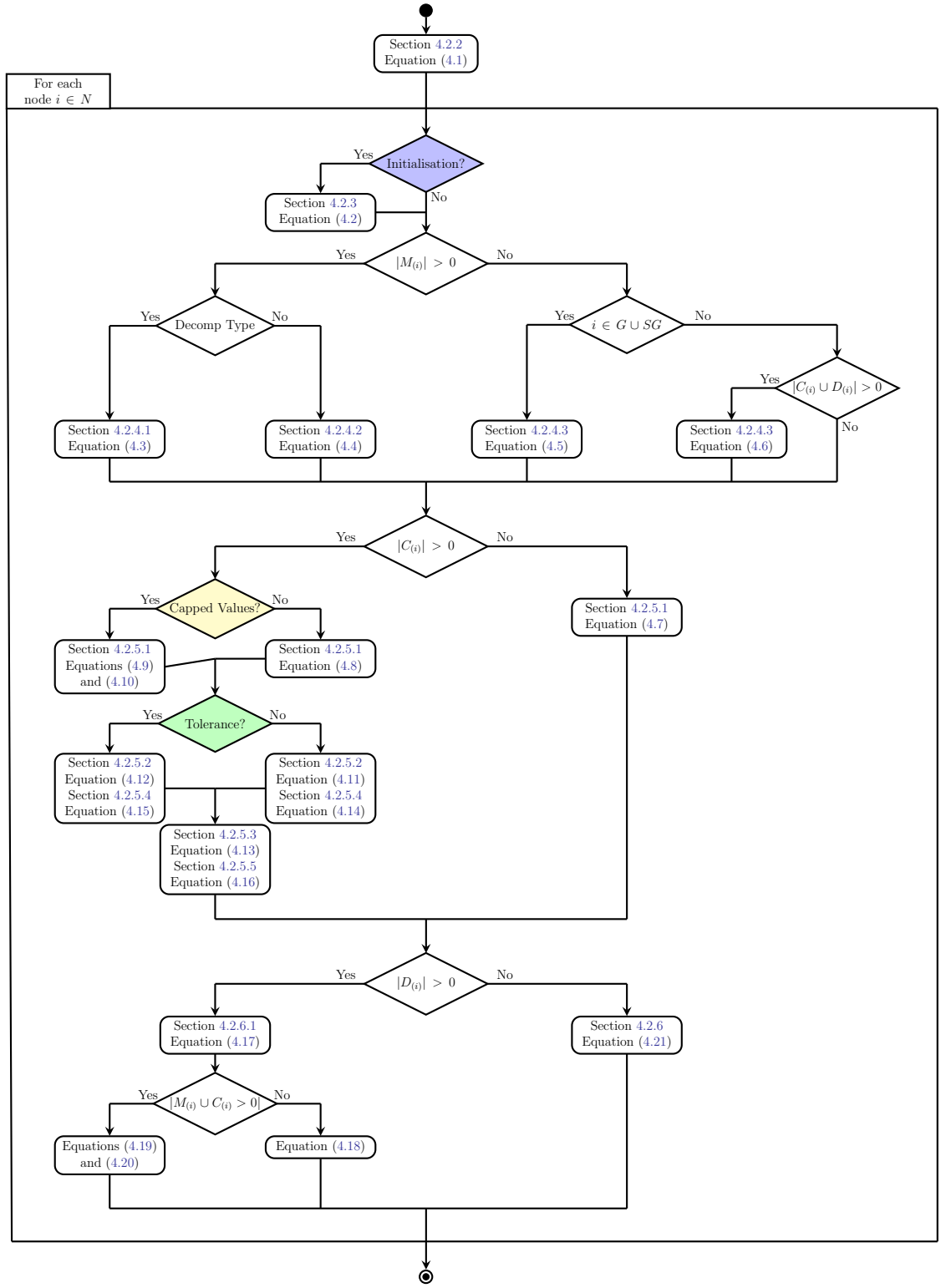


Figure 4.1: Flowchart for use in converting the Goal-oriented Requirements Language schema to a model.

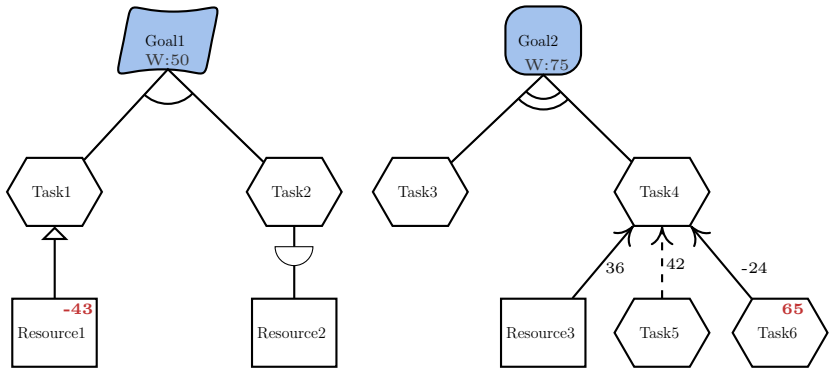


Figure 4.2: A small GRL graph used to show the generation of the optimisation model based on the optimisation schema.

Notation	**Explanation**	**Remarks**
$N = G \cup SG \cup T \cup R$	The sets G, SG, T, R respectively represent the goals, softgoals, tasks and resources.	The generic notation N is used if the node type is irrelevant. Nodes are typically detonated by i, j etc ...
$E = decomp \cup contrib \cup dep$	The set of directed arcs, *decomp* — relationships defined as Decomposition or Means-End *contrib* — relationships defined as Contribution or Correlation *dep* — relationships defined as Dependency	For an arc (i, j) between nodes i and j, node i is considered the parent and node j is considered the child.

Table 4.3: The defined graph structure used in GRL models.

4.2.1 Graph Notation of the Optimisation Schema

The optimisation schema for GRL is presented using the directed graph, $Graph = N, E, A$. Table 4.3 defines the elements of the graph, including notes on how the elements will be referenced during the schema presentation.

Additionally, table 4.4 presents the variables and constants that define every node in the graph. These constants form the decision points of the schema — as seen in figure 4.1 — while the variables $mVal_{(i)}$, $cVal_{(i)}$, and $dVal_{(i)}$ form the backbone of the constraints. All nodes are required to have these variables, even if they are

Variable	Range	Definition	Defined
$mVal_{(i)}$	$(-100, 100)$	The value given to node i from decomposition and means-end relationships. For appropriate nodes this value is assigned by the optimisation process.	Equations (4.2) and (4.6), equations (4.3) and (4.4)
$cVal_{(i)}$	$(-100, 100)$	The value of node i based on $mVal_{(i)}$ and any contribution relationships.	Equation (4.16)
$dVal_{(i)}$	$(-100, 100)$	The final value assigned to node i based on $cVal_{(i)}$ and any dependency relationships.	Equations (4.17) and (4.19)
$imp_{(i)}$	$(0, 100)$	The importance value of node i	Constant
$M_{(i)}$		The set of nodes related to node i by decomposition or means-end relationships. Where node i is the parent.	Constant
$C_{(i)}$		The set of nodes related to node i by contribution relationships. Where node i is the parent.	Constant
$D_{(i)}$		The set of nodes related to node i by dependency relationships. Where node i is the parent.	Constant

Table 4.4: Variables and input constants defined for every node i where $i \in N$.

lacking the associated relationships because they are used to calculate the next value. Hence, if one was missing, an optimisation solver *could* use the variable to create an ideal solution when one does not, in actuality, exist.

The remaining part of this section will define the constraints needed for the schema, additional variables and constants are introduced on an "as needed" basis. To aid in readability, the constraints are presented in the same order as the flow chart, and the objective function is presented first to provide context.

Options	Equation	Generated Constraints
All	(4.1)	$\max\left((0.5 \times dVal_{Goal1}) + (0.75 \times dVal_{Goal2})\right)$

Table 4.5: The objective function generated for the *demonstration graph* presented in figure 4.2

4.2.2 Objective Function

The optimal solution of a given graph occurs when all nodes with an importance rating achieve their maximum value, that is $dVal_{(i)} = 100$. Equation (4.1) presents an objective function based on maximising this goal; by including the importance value of each node the optimisation correctly accounts for the weighting, allowing nodes with a higher importance to have a critical effect. The variable $imp_{(i)}$ is treated as a fraction of 100 as node scores are representative of a percentage.

$$\max \sum_{i \in N} \left(\frac{imp_{(i)}}{100} \cdot dVal_{(i)} \right) \tag{4.1}$$

Demonstration Graph Table 4.5 presents the objective function that is generated for the *demonstration graph*. Technically all nodes $dVal$ could be included with a coefficient of 0, but this would unnecessarily complicate the presentation of the constraint. This objective function is generated regardless of any evaluation options selected.

Variable	Range	Definition	Defined
$default_{(i)}$	$(-100, 100)$	The default value specified by the user. Applied only if node i is a leaf node and the initialisation option is selected	Constant

Table 4.6: Input constants required to implement the node initialisation option.

4.2.3 Node Initialisation

As previously mentioned, the optimisation schema allows for nodes to be initialised by the user. However, only leaf nodes may be initialised without potentially causing an infeasible model. Node i is defined as a leaf node when it has no children, that is $|M_{(i)} \cup C_{(i)} \cup D_{(i)}| = 0$. Equation (4.2) handles the initialisation by utilising the input constant $default_{(i)}$, as defined in table 4.6, if the user does not provide a value — $default_{(i)} = NaN$ — then the constraint is not generated.

$$\text{Given } |M_{(i)}| = 0 \text{ and } |C_{(i)}| = 0 \text{ and } |D_{(i)}| = 0 \text{ and } default_{(i)} \neq NaN \text{ define:}$$
$$mVal_{(i)} = default_{(i)} \tag{4.2}$$

Demonstration Graph The constraint presented in equation (4.2) is only generated if the *Initialisation* evaluation option is selected. Two of the leaf nodes have been provided with initialisation values, each of these nodes requires a separate constraint as shown in Table 4.7.

Option	Equation	Generated Constraints
Init	(4.2)	$mVal_{Resource1} = -43$
Init	(4.2)	$mVal_{Task6} = 65$

Table 4.7: The initialisation constraints generated for the *demonstration graph* presented in figure 4.2, these constraints are only generated if *Initialisation* is selected as an evaluation option.

Variable	Range	Definition	Defined
$m_{(i,j)}$	$(0, 200)$	The auxiliary variable $m_{(i,j)}$ defines the difference between the decomposition value $mVal_{(i)}$ of the parent node i and the final value $dVal_{(j)}$ of the child node j	Equations (4.3) and (4.4)
$m'_{(i,j)}$	$(0, 200)$	The auxiliary variable $m'_{(i,j)}$ forces $m_{(i,j)}$ to be zero for at least one child node j. This ensures the correct value is assigned to $mVal_{(i)}$.	Equations (4.3) and (4.4)

Table 4.8: Variables required to calculate decomposition and means-end relationships

4.2.4 Decomposition and Means-End Relationships

The first step in building the optimisation model is to define $mVal_{(i)}$, this is the variable that the optimisation process will use to assign a value to a given node. However, there are several limitations on this variable. Firstly, if node i has any decomposition or means-end children, that is, $|M_{(i)}| > 0$ then the value of $mVal_{(i)}$ becomes dependant on these children. Section 4.2.4.1 defines the constraints necessary to model an *AND* relationship, while section 4.2.4.2 defines the constraints necessary for an *OR* or *means-end* relationship. Secondly, if a node has no decomposition relationships, but is not a leaf-node, that is, $|C_{(i)}| \cup |D_{(i)}| > 0$, then the final value is dependant on these relationships. Section 4.2.4.3 defines the constraints necessary to implement this reliance. Finally, a goal or softgoal cannot be given a value without satisfactory evidence, that is, direct children or node initialisation, this situation is also covered by section 4.2.4.3. Table 4.8 defines the additional variables required by the specified subsections to calculate $mVal_{(i)}$.

4.2.4.1 AND Decomposition:

An *AND* decomposition relationship implies that a parent can only be satisfied if all its children are satisfied. This means the parent node i cannot take a decomposition value ($mVal_{(i)}$) higher than any one of its children's final values ($dVal_{(j)}$), where

Option	Equation	Generated Constraints
ALL	(4.3a)	$mVal_{Goal1} + m_{Goal1,Task1} = dVal_{Task1}$
ALL	(4.3a)	$mVal_{Goal1} + m_{Goal1,Task2} = dVal_{Task2}$
ALL	(4.3d)	$m'_{Goal1,Task1} + m'_{Goal1,Task2} \geq 1$

Table 4.9: The constraints generated for the *demonstration graph* in order to model the AND decomposition.

node j is in the set $M_{(i)}$. Equation (4.3) applies this restriction by forcing $mVal_{(i)}$ to take the minimum value of it's children nodes.

For every node j in $M_{(i)}$ where the relation is AND define:

$$mVal_{(i)} + m_{(i,j)} = dVal_{(j)} \tag{4.3a}$$

$$m_{(i,j)} \text{ and } m'_{(i,j)} \geq 0 \tag{4.3b}$$

$$m_{i,j} \cdot m'_{(i,j)} = 0 \tag{4.3c}$$

Additionally, define the following constraint

$$\sum_{j \in M_i} \left(m'_{(i,j)}\right) \geq 1 \tag{4.3d}$$

Demonstration Graph Equations (4.3b) and (4.3c) are, as before, modelled as variable limits and special ordered sets rather than constraints. The demonstration graph only has a single AND relationship, so only one set of constraints are generated, as shown in table 4.9

4.2.4.2 OR and Means-End Decomposition:

An *OR* decomposition relationship implies that a parent is satisfied if any of its children are satisfied. This means the parent node i should take the same decomposition value ($mVal_{(i)}$) as its highest scoring child ($dVal_{(j)}$) where node j is in the set $M_{(i)}$. Equation (4.4) applies this restriction by forcing $mVal_{(i)}$ to take the

Option	Equation	Generated Constraints
ALL	(4.4a)	$mVal_{Goal2} + m_{Goal2,Task3} = dVal_{Task3}$
ALL	(4.4a)	$mVal_{Goal2} + m_{Goal2,Task4} = dVal_{Task4}$
ALL	(4.4d)	$m'_{Goal2,Task3} + m'_{Goal2,Task4} \geq 1$
ALL	(4.4a)	$mVal_{Task1} + m_{Task1,Resource1} = dVal_{Resource1}$
ALL	(4.4d)	$m'_{Task1,Resource1} \geq 1$

Table 4.10: The constraints generated for the *demonstration graph* in order to model the OR decomposition.

maximum value of it's children nodes.

For every node j in $M_{(i)}$ where the relation is OR or Means-End define:

$$mVal_{(i)} - m_{(i,j)} = dVal_{(j)} \tag{4.4a}$$

$$m_{(i,j)} \text{ and } m'_{(i,j)} \geq 0 \tag{4.4b}$$

$$m_{i,j} \cdot m'_{(i,j)} = 0 \tag{4.4c}$$

Additionally, define the following constraint

$$\sum_{j \in M_i} \left(m'_{(i,j)}\right) \geq 1 \tag{4.4d}$$

Demonstration Graph Table 4.10 shows the constraints generated and added to the optimisation model in order to implement the single OR relationship in the demonstration graph. Additionally, it shows the constraints generated for the means-end relationship.

4.2.4.3 No Decomposition or Means-End Relationships

The purpose of optimisation is to determine which tasks should be undertaken, and resources supplied, in order to ensure that nodes of importance — $imp_{(i)} > 0$ — achieve their highest possible score. Logically, this implies that the optimisation process should not be able to assign a value to a goal or softgoal unless suitable evidence that such a value exists has been provided, either in the form of children, or initialisation. Equation (4.5) enforces this by assigning $mVal_{(i)}$ a value of 0,

Option	Equation	Generated Constraints
ALL	(4.6)	$mVal_{Task2} = 0$
ALL	(4.6)	$mVal_{Task4} = 0$

Table 4.11: The constraints generated for the *demonstration graph* in order to handle non-leaf nodes with no decomposition children.

where node i is a goal or softgoal and also a leaf node.

$$\begin{gathered}\text{For a node } i \text{ where Equation (4.2) has not been defined, and} \\ \text{Given } |M_{(i)}| = 0 \text{ and } i \in G \cup SG \text{ define:} \\ mVal_{(i)} = 0 \end{gathered} \tag{4.5}$$

The final restriction of the optimisation process is that it should *only* assign a value to a leaf node. If this restriction is not applied, then the optimisation can arbitrarily assign a value to the decomposition variable — $mVal_{(i)}$ — of any node with $|M_{(i)}| = 0$.

$$\begin{gathered}\text{Given } |M_{(i)}| = 0 \text{ and } |C_{(i)} \cup D_{(i)}| > 0 \text{ define:} \\ mVal_{(i)} = 0 \end{gathered} \tag{4.6}$$

Demonstration Graph Table 4.11 shows the constraints generated and added to the optimisation model based on equation (4.6). Equation (4.5) does not apply to the *demonstration graph* as there are no goals or softgoals that are leaf-nodes.

4.2.5 Contribution and Correlation Relationships:

Contribution and correlation relationships are grouped together for the purpose of the optimisation schema, as mathematically they have the same effect and the difference is conceptual. Calculating the effect of contribution and correlation relationships is similar to calculating impacts in chapter 3, that is, it is feasible that the

Step	Purpose	Section	Equations
Initial Value	Calculate the total impact contribution and correlation relationships have.	4.2.5.1	(4.7)–(4.10)
Determine Lower limit	If applicable — determine the lower tolerance value.	4.2.5.2.	(4.11) – (4.12)
Apply Lower Cap	Apply the lower limit to the raw score, score is now in the range $(negTol, \infty)$.	4.2.5.3	(4.13)
Determine Upper Limit	If applicable — determine the upper tolerance value.	4.2.5.4	(4.14) – (4.15)
Apply Upper Cap	Apply the lower limit to the raw score, score is now in the range $(negTol, posTol)$.	4.2.5.5	(4.16)

Table 4.12: The process used to ensure a valid calculation of contribution and correlation relationships.

sum total of these relationships will exceed the lower or upper limit of a node's value[2]. As before, numerous equations are required to handle this situation, table 4.12 elucidates these steps and the responsible subsections and equations.

Table 4.13 introduces the new variables and constants needed to define these constraints. Special attention should be paid to the variables $cVal2_{(i)}$, $cVal1_{(i)}$, and $cVal_{(i)}$, as the schema utilises them to simulate a procedural calculation.

4.2.5.1 Initial Calculation:

In order to calculate the contribution and correlation scores, a starting point is required. As shown in the flow chart (figure 4.1), there are three possible equations we can use, based on whether contribution or correlation relationships exist, and then on the chosen evaluation method.

[2]By default this schema uses the values -100 and 100 as the lower and upper limit respectively. These values represent -100%, and 100%, node satisfaction

Variable	Range	Definition	Defined
$cVal2_{(i)}$	$(-\infty, \infty)$	The initial value calculated from contribution and correlation relationships and including the previous $mVal_{(i)}$ value.	Equations (4.8) and (4.10)
$cVal1_{(i)}$	$(-100, \infty)$	The result of applying the lower limit to $cVal2_{(i)}$	Equation (4.13b)
$capVal_{(i)}$	$(0, \infty)$	An intermediary value calculated by limiting the final $dVal_{(i)}$ or each node to 0.	Equation (4.9b)
$x_{(i)}$ $x'_{(i)}$	$(0, 100)$	The auxiliary variables $x'_{(i)}$(the deficit) and $x_{(i)}$(the surplus) quantify the difference between $dVal_{(i)}$ and the limit 0.	Equation (4.9)
$nT'_{(i)}$ $pT'_{(i)}$	$(0, 200)$	The auxiliary variables $nT'_{(i)}$(the deficit) and $pT'_{(i)}$(the surplus) quantify the difference between $mVal_{(i)}$ and $inputNTol_{(i)}$.	Equation (4.12)
$nTol_{(i)}$	$(-100, 0)$	The negative tolerance for node i based on the $inputNTol_{(i)}$ and the value of $mVal_{(i)}$.	Equation (4.12b)
$tol_{(i)}$	$(0, 100)$	The positive tolerance for node i based on the $inputTol_{(i)}$ and the value of $mVal_{(i)}$	Equation (4.15b)
$n'_{(i)}$ $p'_{(i)}$	$(0, \infty)$	The auxiliary variables $n'_{(i)}$(the deficit) and $p'_{(i)}$(the surplus) quantify the difference between $cVal2_{(i)}$ and the lower limit as represented by $nTol_{(i)}$.	Equation (4.13)
$nT_{(i)}$ $pT_{(i)}$	$(0, 200)$	The auxiliary variables $nT_{(i)}$(the deficit) and $pT_{(i)}$(the surplus) quantify the difference between $mVal_{(i)}$ and $inputTol_{(i)}$.	Equation (4.15)
$n_{(i)}$ $p_{(i)}$	$(0, \infty)$	The auxiliary variables $n_{(i)}$(the deficit) and $p_{(i)}$(the surplus) quantify the difference between $cVal1_{(i)}$ and the upper limit as represented by $tol_{(i)}$.	Equation (4.16)
$inputNTol_{(i)}$	$(-100, 0)$	The negative tolerance assigned to node i as specified by the user.	Constant
$inputTol_{(i)}$	$(0, 100)$	The positive tolerance assigned to node i as specified by the user.	Constant

Table 4.13: Variables required to calculate contribution relationships

Option	Equation	Generated Constraints
ALL	(4.7)	$cVal_{Goal1} = mVal_{Goal1}$
ALL	(4.7)	$cVal_{Goal2} = mVal_{Goal2}$
ALL	(4.7)	$cVal_{Task1} = mVal_{Task1}$
ALL	(4.7)	$cVal_{Task2} = mVal_{Task2}$
ALL	(4.7)	$cVal_{Task3} = mVal_{Task3}$
ALL	(4.7)	$cVal_{Task5} = mVal_{Task5}$
ALL	(4.7)	$cVal_{Task6} = mVal_{Task6}$
ALL	(4.7)	$cVal_{Resource1} = mVal_{Resource1}$
ALL	(4.7)	$cVal_{Resource2} = mVal_{Resource2}$
ALL	(4.7)	$cVal_{Resource3} = mVal_{Resource3}$

Table 4.14: The constraints generated for the *demonstration graph* in order to handle non-leaf nodes with no decomposition children.

No Contribution Children: The first possibility is that a parent node i has no children with a *contrib* relationship, in this case the equation (4.7) forces $cVal_{(i)}$ to take the value of $mVal_{(i)}$ and carry it through to the dependency calculation. If this equation is used then most of the variables defined in table 4.13 are meaningless for node i.

$$\begin{aligned} &\text{If } |C_{(i)}| \leq 0 \\ &cVal_{(i)} = mVal_{(i)} \end{aligned} \tag{4.7}$$

Demonstration Graph Only one node in the *demonstration graph* has contribution or correlation children, the remaining nodes simple carry over their $mVal$, table 4.14 presents the necessary constraints.

Basic Contribution Score: The constraint modelled by equation (4.8) shows the basic method of calculating the effect from contribution relationships. As mentioned above, there are two possible methods of handling this calculation — the yellow decision node in the control flow graph — this constraint models traditional approach of denying a negative relationship. Additionally, the constraint also includes the

previous decomposition value ($mVal_{(i)}$), as in equation (4.7).

$$cVal2_{(i)} = \sum_{\substack{j \in C_{(i)} \\ (i,j) \in contrib}} \left(dVal_{(j)} \cdot \frac{contrib_{(i,j)}}{100} \right) + mVal_{(i)} \tag{4.8}$$

Capped Contribution Score: The alternative to the previous constraint is to only model the effect a child node has if there is evidence that it has been achieved. To implement this logic, the final value of a node $dVal_{(i)}$ must be capped so it falls in the range $(0, 100)$. Therefore, equation (4.9) must be specified for all nodes in the graph.

$$dVal_{(i)} + x'_{(i)} - x_{(i)} = 0 \tag{4.9a}$$

$$capVal_{(i)} = dVal_{(i)} + x'_{(i)} \tag{4.9b}$$

$$x'_{(i)} \text{ and } x_{(i)} \geq 0 \tag{4.9c}$$

$$x'_{(i)} \cdot x_{(i)} = 0 \tag{4.9d}$$

Based on $capVal_{(i)}$, equation (4.10) calculates the value of a node based only on contribution relationships of children with non-negative values, that is, evidence of their existence.

$$cVal2_{(i)} = \sum_{\substack{j \in C_{(i)} \\ (i,j) \in contrib}} \left(capVal_{(j)} \cdot \frac{contrib_{(i,j)}}{100} \right) + mVal_{(i)} \tag{4.10}$$

Demonstration Graph Table 4.15 presents the constraints necessary to implement capped values, two constraints must be generated for each node as shown in the first two sections of the table. The final constraint *replaces* the original calculation of $cVal2_{Task4}$.

Option	Equation	Generated Constraints
¬Capped	(4.8)	$cVal2_{Task4} = (0.36 \times dVal_{Resource3}) + (0.42 \times dVal_{Task5}) + (-0.24 \times dVal_{Task6}) + mVal_{Task4}$
Capped	(4.9a)	$dVal_{Goal1} + x'_{Goal1} - x_{Goal1} = 0$
Capped	(4.9a)	$dVal_{Goal2} + x'_{Goal2} - x_{Goal2} = 0$
Capped	(4.9a)	$dVal_{Task1} + x'_{Task1} - x_{Task1} = 0$
Capped	(4.9a)	$dVal_{Task2} + x'_{Task2} - x_{Task2} = 0$
Capped	(4.9a)	$dVal_{Task3} + x'_{Task3} - x_{Task3} = 0$
Capped	(4.9a)	$dVal_{Task4} + x'_{Task4} - x_{Task4} = 0$
Capped	(4.9a)	$dVal_{Task5} + x'_{Task5} - x_{Task5} = 0$
Capped	(4.9a)	$dVal_{Task6} + x'_{Task6} - x_{Task6} = 0$
Capped	(4.9a)	$dVal_{Resource1} + x'_{Resource1} - x_{Resource1} = 0$
Capped	(4.9a)	$dVal_{Resource2} + x'_{Resource2} - x_{Resource2} = 0$
Capped	(4.9a)	$dVal_{Resource3} + x'_{Resource3} - x_{Resource3} = 0$
Capped	(4.9b)	$capVal_{Goal1} = dVal_{Goal1} + x'_{Goal1}$
Capped	(4.9b)	$capVal_{Goal2} = dVal_{Goal2} + x'_{Goal2}$
Capped	(4.9b)	$capVal_{Task1} = dVal_{Task1} + x'_{Task1}$
Capped	(4.9b)	$capVal_{Task2} = dVal_{Task2} + x'_{Task2}$
Capped	(4.9b)	$capVal_{Task3} = dVal_{Task3} + x'_{Task3}$
Capped	(4.9b)	$capVal_{Task4} = dVal_{Task4} + x'_{Task4}$
Capped	(4.9b)	$capVal_{Task5} = dVal_{Task5} + x'_{Task5}$
Capped	(4.9b)	$capVal_{Task6} = dVal_{Task6} + x'_{Task6}$
Capped	(4.9b)	$capVal_{Resource1} = dVal_{Resource1} + x'_{Resource1}$
Capped	(4.9b)	$capVal_{Resource2} = dVal_{Resource2} + x'_{Resource2}$
Capped	(4.9b)	$capVal_{Resource3} = dVal_{Resource3} + x'_{Resource3}$
Capped	(4.10)	$cVal2_{Task4} = (0.36 \times capVal_{Resource3}) + (0.42 \times capVal_{Task5}) + (-0.24 \times capVal_{Task6}) + mVal_{Task4}$

Table 4.15: The constraints generated for the *demonstration graph* in order to calculate capped contribution score.

4.2.5.2 Determine Negative Tolerance

No Tolerance: The constraints in this section relate to the tolerance evaluation method — the green decision node. If a tolerance calculation is *not* to be used, then the constraint presented by equation (4.11) should be applied. This constraint assigns the default value -100 to the variable $nTol_{(i)}$, as the variable is required in later constraints.

$$nTol_{(i)} = -100 \tag{4.11}$$

Tolerance Evaluation: Alternatively, if a tolerance evaluation method is applied, then the constraints presented in equation (4.12) are required. These constraints are based on the input constant $inputNTol_{(i)}$, the value of which is assigned by the user, before its use the value of this constant may be modified if any of the contribution relationships have a stronger affect. Equations (4.12a) and (4.12b) are required to determine if the node is allowed to take a value of -100 due to the previous value imparted from decomposition relationships ($mVal_{(i)}$).

$$mVal_{(i)} + nT'_{(i)} - pT'_{(i)} = inputNTol_{(i)} \tag{4.12a}$$

$$nTol_{(i)} = mVal_{(i)} - pT'_{(i)} \tag{4.12b}$$

$$nT'_{(i)} \cdot pT'_{(i)} = 0 \tag{4.12c}$$

Demonstration Graph Table 4.16 shows the constraints that are generated in order to determine the lower tolerance. The two sets of constraints are mutually exclusive, they both result in a calculation for $nTol$.

4.2.5.3 Apply Lower Limit

Now that $nTol_{(i)}$ has been defined, equation (4.13) can use this variable to apply a lower limit to the contribution value $cVal2_{(i)}$. A lower limit needs to be applied

Option	Equation	Generated Constraints
¬Tol	(4.11)	$nTol_{Task4} = -100$
Tol	(4.12a)	$mVal_{Task4} + nT'_{Task4} - pT'_{Task4} = inputNTol_{Task4}$
Tol	(4.12b)	$nTol_{Task4} = mVal_{Task4} - pT'_{Task4}$

Table 4.16: The constraints generated to determine the negative tolerance that should be applied to contribution and correlation relationships.

Option	Equation	Generated Constraints
ALL	(4.13a)	$cVal2_{Task4} + n'_{Task4} - p'_{Task4} = nTol_{Task4}$
ALL	(4.13b)	$cVal1_{Task4} = cVal2_{Task4} + n'_{Task4}$

Table 4.17: The constraints generated to ensure the contribution and correlation relationships do not exceed the lower bound.

because, as previously stated, multiple relationships can result in a value lower than -100. Due to the previous calculation of $nTol_{(i)}$, these constraints do not need to be modified if tolerance evaluation is used.

$$cVal2_{(i)} + n'_(i) - p'_{(i)} = nTol_{(i)} \tag{4.13a}$$

$$cVal1_{(i)} = cVal2_{(i)} + n'_{(i)} \tag{4.13b}$$

$$n'_{(i)} \text{ and } p'_{(i)} \geq 0 \tag{4.13c}$$

$$n'_{(i)} \cdot p'_{(i)} = 0 \tag{4.13d}$$

Demonstration Graph Table 4.17 shows the two constraints that are generated in order to apply the lower limit.

4.2.5.4 Determine Positive Tolerance:

The constraints presented here are the polar opposite of those in section 4.2.5.2, that is, they calculate the positive tolerance $tol_{(i)}$.

Option	Equation	Generated Constraints
¬Tol	(4.14)	$tol_{Task4} = 100$
Tol	(4.15a)	$mVal_{Task4} + nT_{Task4} - pT_{Task4} = inputTol_{Task4}$
Tol	(4.15b)	$tol_{Task4} = mVal_{Task4} + nT_{Task4}$

Table 4.18: The constraints generated to determine the positive tolerance that should be applied to contribution and correlation relationships.

No Tolerance: In situations where tolerance evaluation is not to be used, equation (4.14) assigns a default value to the variable $tol_{(i)}$ so the variable may be used in later constraints.

$$tol_{(i)} = 100 \tag{4.14}$$

Tolerance Evaluation: Similar to equation (4.12), equation (4.15) determines if a node can take a value of 100 due to the effect of decomposition relationships.

$$mVal_{(i)} + nT_{(i)} - pT_{(i)} = inputTol_{(i)} \tag{4.15a}$$

$$tol_{(i)} = mVal_{(i)} + nT_{(i)} \tag{4.15b}$$

$$nT_{(i)} \cdot pT_{(i)} = 0 \tag{4.15c}$$

Demonstration Graph Table 4.18 shows the constraints that are generated in order to determine the upper tolerance. The two sets of constraints are mutually exclusive, they both result in a calculation for tol.

4.2.5.5 Apply Upper Limit

The same process used to apply the lower limit is used to apply the upper limit. Equation (4.16) uses the positive tolerance $tol_{(i)}$ to apply the upper limit to $cVal1_{(i)}$,

Option	Equation	Generated Constraints
ALL	(4.13a)	$cVal1_{Task4} + n_{Task4} - p_{Task4} = tol_{Task4}$
ALL	(4.13b)	$cVal_{Task4} = cVal1_{Task4} - p_{Task4}$

Table 4.19: The constraints generated to ensure the contribution and correlation relationships do not exceed the upper bound.

the variable representing the nodes score *after* application of the lower limit.

$$cVal1_{(i)} + n_(i) - p_{(i)} = tol_{(i)} \tag{4.16a}$$

$$cVal_{(i)} = cVal1_{(i)} - p_{(i)} \tag{4.16b}$$

$$n_{(i)} \text{ and } p_{(i)} \geq 0 \tag{4.16c}$$

$$n_{(i)} \cdot p_{(i)} = 0 \tag{4.16d}$$

Demonstration Graph Table 4.19 shows the two constraints that are generated in order to apply the upper limit.

4.2.6 Dependency Relationships:

The final relationship that needs to be addressed is *dependencies*, table 4.20 introduces the variables necessary for this task. There are several steps necessary for this calculation, first, the initial effect of the dependency relationships must be determined, as in section 4.2.6.1. Then the effect of any previous relationships must be taken into account, as explained in section 4.2.6.2. Finally, a separate case must be made for those nodes that do not have any dependency relationships, as in section 4.2.6.3.

4.2.6.1 Initial Calculation

The nature of dependency relationships function such that node i may not take a value higher than any node it depends upon. Equation (4.17) uses the variables $d_{(i,j)}$

Variable	Range	Definition	Defined
$d_{(i,j)}$	$(0, 200)$	The auxiliary variable $d_{(i,j)}$ defines the difference between the final value $dVal_{(i)}$ of the parent node i and the final value $dVal_{(j)}$ of the child node j	Equation (4.17)
$d'_{(i,j)}$	$(0, 200)$	The auxiliary variable $d'_{(i,j)}$ forces $d_{(i,j)}$ to be zero for at least one child node j. This ensures the correct value is assigned to $dVal_{(i)}$.	Equation (4.17)
$c_{(i)}$	$(0, 200)$	The equivalent variable of $d_{(i,j)}$, $c_{(i)}$ defines the difference between the final value $dVal_{(i)}$ of the parent node i and the value determined by contribution/correlation $cVal_{(i)}$, used only if any non-dependency relationships are present.	Equation (4.19)
$c'_{(i)}$	$(0, 200)$	The equivalent variable $d'_{(i,j)}$, used only if any no dependency relationships are present.	Equation (4.19)

Table 4.20: Variables required to calculate dependency relationships

and $d'_{(i,j)}$ to determine the lowest value in the set $D_{(i)}$, this constraint functions in the same manner as in section 4.2.4.1.

For every node j in $D_{(i)}$ define:

$$dVal_{(i)} + d_{(i,j)} = dVal_{(j)} \tag{4.17a}$$

$$d_{(i,j)} \text{ and } d'_{(i,j)} \geq 0 \tag{4.17b}$$

$$d_{(i,j)} \cdot d'_{(i,j)} = 0 \tag{4.17c}$$

Equation (4.18) is used to ensure that $dVal_{(i)}$ takes the correct value, though, this constraint should only be defined if no decomposition or correlation relationships exist.

If $|M_{(i)} \cup C_{(i)}| = 0$ define:

$$\sum_{j \in M_i} \left(d'_{(j)}\right) = 1 \tag{4.18}$$

Option	Equation	Generated Constraints
ALL	(4.17a)	$dVal_{Task2} + d_{(Task2,Resource1)} = dVal_{Resource1}$
ALL	(4.17a)	$d'_{(Task2,Resource1)} = 1$

Table 4.21: The constraints generated to account for dependency relationships.

Demonstration Graph Table 4.21 shows the two constraints that are generated in order to account for the dependency relationship. In this case as there is only a single dependency child it essentially applies a maximum limit on $dVal_{Task2}$.

4.2.6.2 Account for previous Relationships

If a variable has any decomposition or contribution relationships, then the value from these relationships must be included in the calculation of $dVal_{(i)}$. Hence, if $|M_{(i)} \cup C_{(i)}| > 0$, then equation (4.19) should be defined in order to add the variable $cVal_{(i)}$ to the calculation of $dVal_{(i)}$.

$$\text{If } |M_{(i)} \cup C_{(i)}| > 0 \text{ define:}$$

$$dVal_{(i)} + c_{(i)} = cVal_{(i)} \tag{4.19a}$$

$$c_{(i)} \text{ and } c'_{(i)} \geq 0 \tag{4.19b}$$

$$c_i \cdot c'_{(i)} = 0 \tag{4.19c}$$

In this situation, equation (4.18) must be replaced with equation (4.20), this constraint functions in the same manner but also allows for the value of $cVal_{(i)}$.

$$\text{If } |M_{(i)} \cup C_{(i)}| > 0 \text{ define:}$$

$$\sum_{j \in M_i} \left(d'_{(j)}\right) + c'_{(i)} = 1 \tag{4.20}$$

Demonstration Graph The demonstration graph has no nodes with two different types of children, so no constraints are generated based on equation (4.19).

Option	Equation	Generated Constraints
ALL	(4.21)	$dVal_{Goal1} = cVal_{Goal1}$
ALL	(4.21)	$dVal_{Goal2} = cVal_{Goal2}$
ALL	(4.21)	$dVal_{Task1} = cVal_{Task1}$
ALL	(4.21)	$dVal_{Task3} = cVal_{Task3}$
ALL	(4.21)	$dVal_{Task4} = cVal_{Task4}$
ALL	(4.21)	$dVal_{Task5} = cVal_{Task5}$
ALL	(4.21)	$dVal_{Task6} = cVal_{Task6}$
ALL	(4.21)	$dVal_{Resource1} = cVal_{Resource1}$
ALL	(4.21)	$dVal_{Resource2} = cVal_{Resource2}$
ALL	(4.21)	$dVal_{Resource3} = cVal_{Resource3}$

Table 4.22: The constraints generated for the *demonstration graph* in order to propagate the final values of nodes with no dependency relationships.

4.2.6.3 No Dependency Relationships

If a node has no dependency relationships, then equation (4.21) ensures the value from $cVal_{(i)}$, is carried through to $dVal_{(i)}$ as the latter variable is used to refer to the nodes final value when specifying the constraints of other nodes. While this effect could be implemented using the constraints in section 4.2.6.2, the superfluous constraints would increase the complexity of the model.

$$dVal_{(i)} = cVal_{(i)} \tag{4.21}$$

Demonstration Graph Only one node in the *demonstration graph* has a dependency child, the remaining nodes simple carry over their $cVal$, table 4.22 presents the necessary constraints.

4.2.7 Actor Calculation

The modelling of actor responsibility is a common practice when dealing with goal graphs. However, actors have no affect on the objective function and hence, their inclusion in the optimisation schema would be redundant. Developers can apply any

actor calculation they desire to the results of the optimisation schema.

4.3 Schema Verification

The previous section developed the optimisation schema for GRL, as figure 4.1 outlined, this schema is significantly larger than that developed in the previous chapter. To ensure that each of the constraints resulted in the desired outcome, they were applied to a series of small test graphs. Initially, only the constraints pertaining to a single relationship were examined, once individual relationships were verified the complexity of the test graphs was increased to incorporate multiple relationships.

The simplicity of the initial graphs resulted in equations that can not only be generated by hand but also *evaluated*, this allowed the effects of the constraints to be clearly observed. By assigning various values to the leaf nodes, these observations were used to ensure that the leaf node values were being correctly propagated to their parents. This was done in two phases, first it was determined if the constraints allowed the correct value to be propagated to the parent. Next *proof by contradiction ([66])* was utilised to ensure that the incorrect child value could not reach the parent, and that the parent could not take any other value. Proof by contradiction was utilised by attempting to propagate a value — either towards the parent, or towards the child — then determining at which point a constraint failed to hold.

Verification of the propagation process proved that parent nodes received correct values from the leaf nodes. The next step was to ensure the objective function resulted in correct assignment of leaf-softgoal values. This was done by creating a simulation similar to that used in section 3.6.3, the results were then altered in various ways, propagated, and the objective value calculated, these values were compared to ensure the simulation had produced the maximum value of the objective function.

This verification process was repeated using various node types and under the different evaluation techniques described in section 4.2. It was important to evaluate the same graph structure with different node types, as tasks/resources are the determinate of the optimisation — the optimisation software may assign them values — whereas goals/softgoals are the objective, and as previously established may only receive a value from their children.

As should be apparent, this verification process often involved applying minute transformations to either the graphs or constraints. Hence, this section provides a single graph for each relationship, a decision that was made to aid in the readability of the results and the understandability of the process. Appendix A expands these demonstrations to include a multitude of graphs and value combinations.

4.3.1 Decomposition — *AND*

The first relationship under examination is decomposition, specifically, the *AND* decomposition. This discussion will be based on equation (4.3) as presented in section 4.2.4.1. As decomposition is the "starting point" of the propagation, there is no previous value to be taken into account.

Figure 4.3 presents a simple three node graph constructed using an *AND* relationship. As discussed in section 4.2.4.1, in an *AND* relationship the parent node takes the minimal value of its children, in this example that means the parent node A must take the value of child node C. The first portion of table 4.23 presents all the constraints generated to correctly propagate this value to the parent; all these constraints hold as true. The remaining portions of the table attempt to propagate an incorrect value. The first incorrect propagation attempts to assign node A the value of node B, the second attempt tries to assign node A a score higher than either child node, and the final attempt tries to assign A a value lower than either of the children. In each of these cases the violated constraint has been highlighted for readability. This table demonstrates that in order to accommodate the various situations, all of the constraints from equation (4.3) are required.

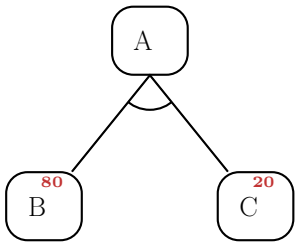


Figure 4.3: Simple three node graph with an *AND* decomposition.

Reference	Child	Constraint	Holds or Breaks
Correct Propagation: $mVal_{(A)} = 20$.			
(4.3a)	B	$20 + 60 = 80$	Holds
(4.3b)	B	$60 \geq 0$	Holds
(4.3c)	B	$60 \cdot 0 = 0$	Holds
(4.3a)	C	$20 + 0 = 20$	Holds
(4.3b)	C	$0 \geq 0$	Holds
(4.3c)	C	$0 \cdot 1 = 0$	Holds
(4.3d)	B, C	$0 + 1 \geq 1$	Holds
Incorrect Propagation: $mVal_{(A)} = 80$.			
(4.3a)	B	$80 + 0 = 80$	Holds
(4.3b)	B	$0 \geq 0$	Holds
(4.3c)	B	$0 \cdot 1 = 0$	Holds
(4.3a)	C	$80 + (-60) = 20$	Holds
(4.3b)	C	$-60 \geq 0$	Breaks
(4.3c)	C	$-60 \cdot 0 = 0$	Holds
(4.3d)	B, C	$1 + 0 \geq 1$	Holds
Incorrect Propagation: $mVal_{(A)} = 100$.			
(4.3a)	B	$100 + (-20) = 80$	Holds
(4.3b)	B	$-20 \geq 0$	Breaks
(4.3c)	B	$-20 \cdot 0 = 0$	Holds
(4.3a)	C	$100 + (-80) = 20$	Holds
(4.3b)	C	$-80 \geq 0$	Breaks
(4.3c)	C	$-80 \cdot 0 = 0$	Holds
(4.3d)	B, C	$0 + 0 \geq 1$	Breaks
Incorrect Propagation: $mVal_{(A)} = -20$.			
(4.3a)	B	$-20 + 100 = 80$	Holds
(4.3b)	B	$100 \geq 0$	Holds
(4.3c)	B	$100 \cdot 0 = 0$	Holds
(4.3a)	C	$-20 + 40 = 20$	Holds
(4.3b)	C	$40 \geq 0$	Holds
(4.3c)	C	$40 \cdot 0 = 0$	Holds
(4.3d)	B, C	$0 + 0 \geq 1$	Breaks

Table 4.23: Constraint combinations used to propagate values through an *AND* decomposition — based on figure 4.3.

4.3.2 Decomposition — *OR*

The next relationship is the *OR* decomposition, based on equation (4.4) as presented in section 4.2.4.2; once again no previous value needs to be accounted for. Figure 4.4 presents a simple three node graph constructed using an *OR* relationship. As discussed in section 4.2.4.2, in an *OR* relationship the parent node takes the maximum value of its children, in this example that means the parent node A must take the value of child node B. The first portion of table 4.24 presents all the constraints generated to correctly propagate this value to the parent; all these constraints hold as true. The remaining portions of the table attempt to propagate an incorrect value. The first incorrect propagation attempts to assign node A the value of node C, the second attempt tries to assign node A a perfect score, and the final attempt tries to assign A a value lower than either of the children. In each of these cases the violated constraint has been highlighted for readability. This table demonstrates that in order to accommodate the various situations, all of the constraints from equation (4.4) are required.

4.3.3 Contribution and Correlation

The next step in the propagation is contribution and correlation relationships as presented in section 4.2.5; at this point $mVal_{(i)}$ — the decomposition value – also needs to be accounted for. The variable $mVal_{(i)}$ is only applied as an additive value in equation (4.8), as such its value is redundant in the application of a lower and upper limit. Hence, to simplify the demonstration, figure 4.5 presents a graph with only contribution relationships. The initial section of table 4.25 presents the constraints generated to correctly propagate the value of the children nodes X and Y to the parent node A. The second part attempts to use equation (4.13) to assign a higher value to $cVal1_{(A)}$ and incidentally $cVal_{(A)}$, at this point the mutual exclusion constraint fails to hold — as highlighted in red. Next, the variable $cVal_{(A)}$ is directly improved, at this point one of the variables must become negative, which violates equation (4.16c) — again highlighted in red.

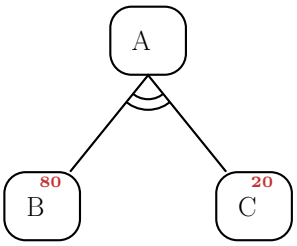


Figure 4.4: Simple three node graph with an *OR* decomposition.

Reference	Child	Constraint	Holds or Breaks
Correct Propagation: $mVal_{(A)} = 80$.			
(4.4a)	B	$80 - 0 = 80$	Holds
(4.4b)	B	$0 \geq 0$	Holds
(4.4c)	B	$0 \cdot 1 = 0$	Holds
(4.4a)	C	$80 - 60 = 20$	Holds
(4.4b)	C	$60 \geq 0$	Holds
(4.4c)	C	$60 \cdot 0 = 0$	Holds
(4.4d)	B, C	$1 + 0 \geq 1$	Holds
Incorrect Propagation: $mVal_{(A)} = 20$.			
(4.4a)	B	$20 - (-60) = 80$	Holds
(4.4b)	B	$-60 \geq 0$	Breaks
(4.4c)	B	$-60 \cdot 0 = 0$	Holds
(4.4a)	C	$20 - 0 = 20$	Holds
(4.4b)	C	$0 \geq 0$	Holds
(4.4c)	C	$0 \cdot 1 = 0$	Holds
(4.4d)	B, C	$0 + 1 \geq 1$	Holds
Incorrect Propagation: $mVal_{(A)} = 100$.			
(4.4a)	B	$100 - 20 = 80$	Holds
(4.4b)	B	$20 \geq 0$	Holds
(4.4c)	B	$20 \cdot 0 = 0$	Holds
(4.4a)	C	$100 - 80 = 20$	Holds
(4.4b)	C	$80 \geq 0$	Holds
(4.4c)	C	$80 \cdot 0 = 0$	Holds
(4.4d)	B, C	$0 + 0 \geq 1$	Breaks
Incorrect Propagation: $mVal_{(A)} = -20$.			
(4.4a)	B	$-20 - (-100) = 80$	Holds
(4.4b)	B	$-100 \geq 0$	Breaks
(4.4c)	B	$-100 \cdot 0 = 0$	Holds
(4.4a)	C	$-20 - (-40) = 20$	Holds
(4.4b)	C	$-40 \geq 0$	Breaks
(4.4c)	C	$-40 \cdot 0 = 0$	Holds
(4.4d)	B, C	$0 + 0 \geq 1$	Breaks

Table 4.24: Constraint combinations used to propagate values through an *OR* decomposition — based on figure 4.4.

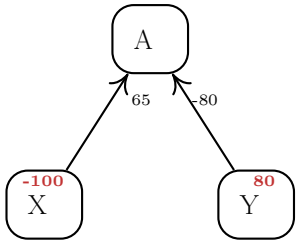


Figure 4.5: Simple three node graph with contribution relationships.

Reference	Constraint	Holds or Breaks
	Correct Propagation: $cVal_{(A)} = -100$.	
(4.8)	$(-100 \cdot 0.65) + (80 \cdot -0.8) = -129$	Holds
(4.13a)	$-129 + 29 - 0 = -100$	Holds
(4.13c)	$29 \geq 0$	Holds
	$0 \geq 0$	Holds
(4.13d)	$29 \cdot 0 = 0$	Holds
(4.13b)	$-129 + 29 = -100$	Holds
(4.16a)	$-100 + 200 - 0 = 100$	Holds
(4.16c)	$200 \geq 0$	Holds
	$0 \geq 0$	Holds
(4.16d)	$200 \cdot 0 = 0$	Holds
(4.16b)	$-100 - 0 = -100$	Holds
	Incorrect Propagation: $cVal1_{(A)} = 100$.	
(4.8)	$(-100 \cdot 0.65) + (80 \cdot -0.8) = -129$	Holds
(4.13a)	$-129 + 229 - 200 = -100$	Holds
(4.13c)	$229 \geq 0$	Holds
	$0 \geq 0$	Holds
(4.13d)	$229 \cdot 200 = 0$	Breaks
(4.13b)	$-129 + 229 = 100$	Holds
	Incorrect Propagation: $cVal_{(A)} = 100$ (let $cVal1_{(A)} = -100$).	
(4.16a)	$-100 + 0 - (-200) = 100$	Holds
(4.16c)	$0 \geq 0$	Holds
	$-200 \geq 0$	Breaks
(4.16d)	$200 \cdot 0 = 0$	Holds
(4.16b)	$-100 - (-200) = 100$	Holds

Table 4.25: Constraint calculations for contribution and correlation relationships based on figure 4.4.

The propagation in table 4.25 uses the basic form of the schema, that is, no tolerance is applied and capped values are not used. These equations can be examined in isolation, as they only result in a different value being assigned to $cVal2_{(A)}$, $nTol_{(A)}$, and $tol_{(A)}$. Based on the same graph as before — figure 4.5 — table 4.26 applies equation (4.9) to the children nodes X and Y, and then uses the capped values to calculate the raw contribution score of node A according to equation (4.10). As before, the first part of the table demonstrates the correct propagation of values. The remaining three sections of the table highlight which constraints are violated should an incorrect value propagation be attempted; for readability, only the equations for the node in question are shown.

The remaining equations, those related to applying a tolerance, are in effect again applying a maximum or minimum calculation, though this time to the variables $inputNTol_{(i)}$, $inputTol_{(i)}$, and $mVal_{(i)}$. The previous discussion has already demonstrated how the constraints used to model a `min` or `max` function cannot be violated. To prevent repetition, these identical constraints will not be provided again.

4.3.4 Dependency Relationships

The constraints used to calculate dependency relationships use the same process as an *AND* relationship to take the minimum value. Hence, the demonstration of this relationship would be identical to section 4.3.1, and has been excluded to prevent redundancy. Appendix A.3 provides various demonstrations of dependency relationships.

Reference	Node	Constraint	Holds or Breaks
Correct Propagation: $capVal_{(X)} = 0$, $capVal_{(Y)} = 80$, $cVal2_{(A)} = -64$.			
(4.9a)	X	$-100 + 100 - 0 = 0$	Holds
(4.9c)	X	$100 \geq 0$	Holds
		$0 \geq 0$	Holds
(4.9d)	X	$100 \cdot 0 = 0$	Holds
(4.9b)	X	$-100 + 100 = 0$	Holds
(4.9a)	Y	$80 + 0 - 80 = 0$	Holds
(4.9c)	Y	$0 \geq 0$	Holds
		$80 \geq 0$	Holds
(4.9d)	Y	$0 \cdot 80 = 0$	Holds
(4.9b)	Y	$80 + 0 = 80$	Holds
(4.10)	A	$(0 \cdot 0.65) + (80 \cdot -0.8) = -64$	Holds
Incorrect Propagation: $capVal_{(X)} = 100$			
(4.9a)	X	$-100 + 200 - 100 = 0$	Holds
(4.9c)	X	$100 \geq 0$	Holds
		$0 \geq 0$	Holds
(4.9d)	Y	$200 \cdot 100 = 0$	Breaks
(4.9b)	X	$-100 + 200 = 100$	Holds
Incorrect Propagation: $capVal_{(X)} = -100$			
(4.9a)	X	$-100 + 0 - (-100) = 0$	Holds
(4.9c)	X	$0 \geq 0$	Holds
		$-100 \geq 0$	Breaks
(4.9d)	X	$100 \cdot 0 = 0$	Holds
(4.9b)	X	$-100 + 0 = -100$	Holds
Incorrect Propagation: $capVal_{(Y)} = 100$			
(4.9a)	Y	$80 + 20 - 100 = 0$	Holds
(4.9c)	Y	$20 \geq 0$	Holds
		$100 \geq 0$	Holds
(4.9d)	Y	$20 \cdot 100 = 0$	Breaks
(4.9b)	Y	$80 + 20 = 80$	Holds

Table 4.26: Constraints used to calculate the capped values of the children nodes in figure 4.5.

4.4 Test Case Application

The previous section demonstrated the ability of the schema constraints to *correctly* propagate values throughout a graph. The small theoretical graphs used for this process were carefully constructed to verify specific constraints and their interactions, however, their size was limited and they bore minimal resemblance to real world situations. This intermediary step of verifying the optimisation schema was necessary as it allowed errors to be identified and corrected, but it did not guarantee a viable schema.

This section assess the viability of the optimisation schema by evaluating real word GRL graphs of increasing complexity. Section 4.4.1 takes the first step of this process by applying the schema to an exemplar from current, established research, while the two remaining subsections apply the optimisation schema to real world situations. Each graph was evaluated multiple times in order to demonstrate the various evaluation options defined in section 4.2 and figure 4.1. Additionally, section 4.4.4 will illustrate the computation time of the various test cases to determine the feasibility of attaining results.

4.4.1 Telecommunication Exemplar

Before a real world situation of considerable size was subjected to the optimisation process, the schema was applied to an exemplar prevalent in current works. The exemplar chosen for this purpose depicts the impact that the location of a new wireless service, and its data, would have on an existing network [43], this exemplar was initially presented in [38].

Figure 4.6 shows this exemplar, with quantitative data, as presented in [43]. The complete exemplar also included actor boundaries, as section 4.2.7 discussed the calculations for actors is independent of the optimisation schema. Hence, actors and their boundaries have been purposefully omitted from the figure in order to improve the readability of the graph. Additionally, any default values in the graph

Nodes	**16**
Goal	3
Softgoal	7
Task	5
Resource	1
Relationships	**16**
Decomposition	4
Correlation/Contribution	10
Dependency	2

Table 4.27: Size breakdown of the telecommunication exemplar.

have been omitted until appropriate, as not all evaluation strategies use these values. Table 4.27 summarises the size of this exemplar based on the number of nodes and relationships. From here on, this exemplar will be referred to as the *Telecommunication Exemplar*.

4.4.1.1 Basic Optimisation Schema

Figure 4.7 shows the results of the basic optimisation schema as applied to the Telecommunication Exemplar, that is, no initialisation, no capped values, and no tolerance. The final value of the objective function for this evaluation was 203.125 of a maximum 350.0, in this smaller exemplar it is easy to see where the bottlenecks occurred.

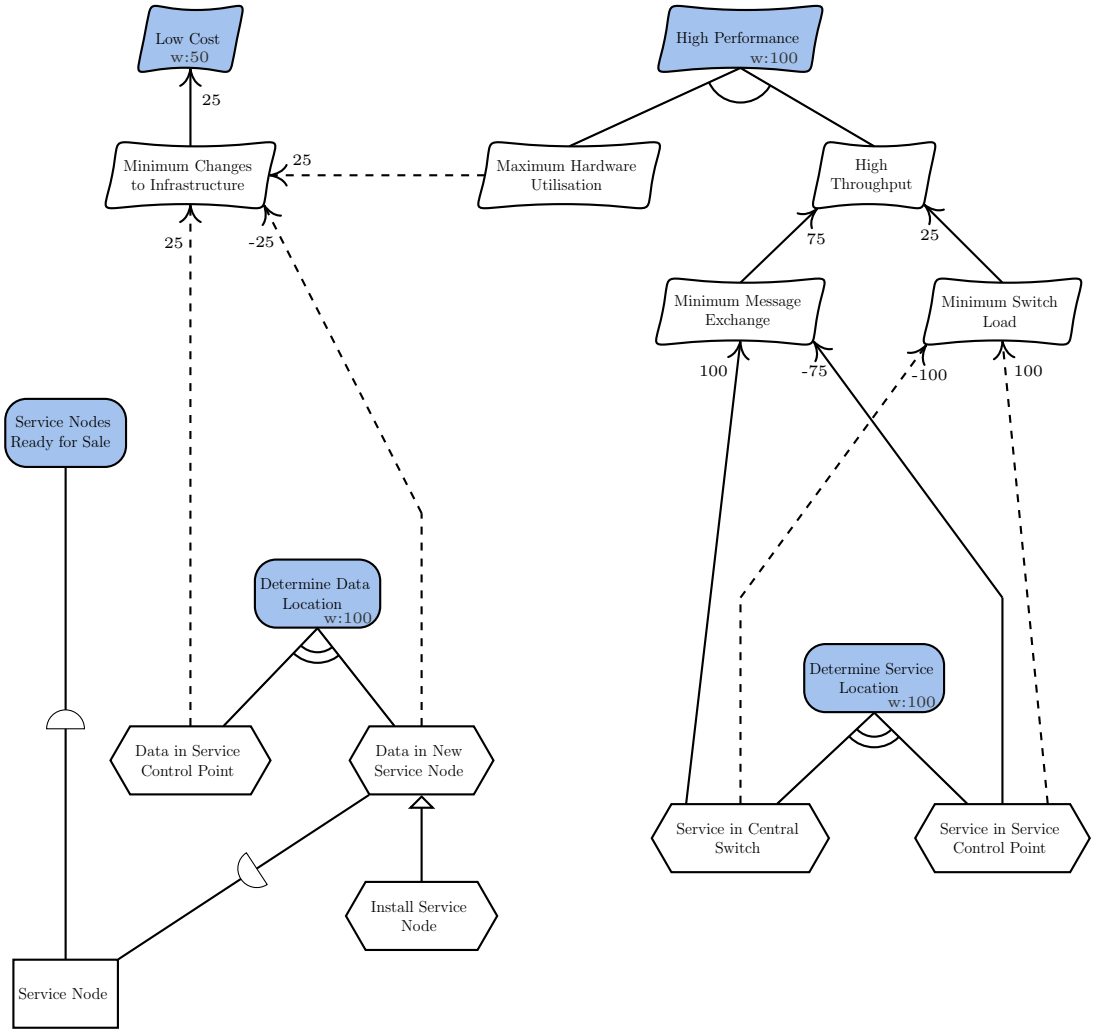


Figure 4.6: GRL Graph describing the impact of the selection of the location of a new wireless service and its data in an existing network.

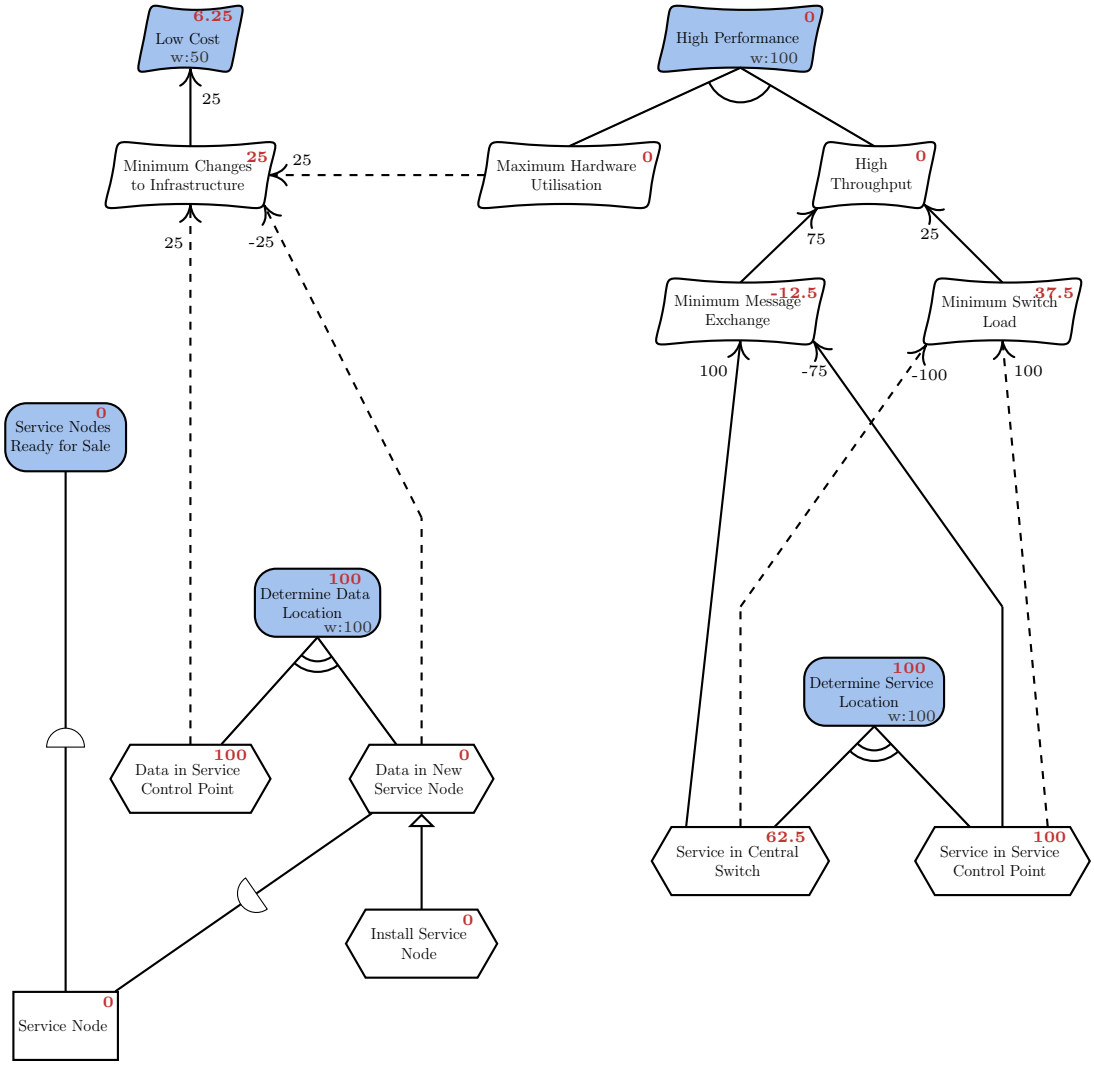


Figure 4.7: Results of the basic optimisation schema as applied to the Telecommunication Exemplar.

Low Cost: *Low Cost* only achieved a score of 6.25. *Low Cost* only has a single child, with a contribution weight of 25, hence, its maximum score is actually 25. The child node *Minimum Changes to Infrastructure* has a potential maximum score of 75, however, the optimisation process may only assign values to *tasks* and *resources*. This means the child node is unable to receive a high value as it required the softgoal *Maximum Hardware Utilisation* and the goal *Service Nodes Ready for Sale* to have non-zero values.

High Performance: *High Performance* is also limited by its child nodes, in this case directly requiring the softgoal *Maximum Hardware Utilisation* to have a non-zero value.

Both of these bottlenecks can be narrowed down to one cause, *goals* or *softgoals* that are leaf-nodes of the graph. This issue can be solved in two ways, the first option would be to provide a complete graph that provides implementation methods for these goals. This approach may not be ideal though, as it would require greatly extending the scope of the graph, and in the case of *Service Nodes Ready for Sale* would extend the scope beyond the limits of the system. The alternative is to provide initialisation values for these nodes, initialisation values allow the incorporation of information that may be infeasible to model, ie: *Service Nodes Ready for Sale*. They also provide an opportunity to explore the potential effects a node value may have on the graph *before* expending resources in the investigation of solutions to the goal, ie: *Maximum Hardware Utilisation*.

4.4.1.2 Optimisation Schema with Initialisation

The basic optimisation schema identified two potential nodes where initialisation may prove useful. *Service Nodes Ready for Sale* is a *goal* node and outside the scope of the system, this node could be provided an initialisation value but it would have a detrimental effect on *Low Cost*. As *Determine Data Location* has already been satisfied there is no need for this goal, hence, it was not given an initialisation value.

The second node *Maximum Hardware Utilisation* would, however, benefit from an initialisation value. Realistically, 100% usage of hardware is unlikely, so an attainable goal of 50% was used. Figure 4.8 shows the results of the optimisation using these initialisation values, as highlighted. This time the value of the objective function was 254.6875. As expected, *Low Cost* only received a marginally increased value. *High Performance* on the other hand now has a value of 50, the highest value it can take given the initialisation applied to an *AND* decomposition child.

At this point, further analysis of this exemplar could take two routes, first, the initialisation values could be altered to determine at which point the value of *High Performance* is solely determined by *High Throughput*. Alternatively, the developers could decide that further investigation into *Maximum Hardware Utilisation* is warranted, this would result in *Maximum Hardware Utilisation* no longer being a leaf node and consequently not taking an initialisation value.

4.4.1.3 Optimisation Schema with Capped Values and Tolerance

The Telecommunication Exemplar is a simplistic example often utilised for discussion as it is easily understandable, this simplicity means the application of capped values or a tolerance approach would be meaningless for the following reasons:

Capped Values: The use of capped values was established largely to enforce the analogy *"two wrongs do not make a right"*. That is to say, if a node has a negative score and a negative contribution to its parent, then its parent should not receive a positive score. In this exemplar, the described situation does not occur. The only impact capped values would impart is *High Throughput* having a value of 75 rather than 50, an increase not propagated to its parent.

Tolerance: Use of a tolerance also has a specific goal, that is, to prevent a node taking a maximum or minimum value by combining several small contributions. In this exemplar, the situation never occurs, the nodes with a value of 100 already have a maximum or minimum contribution to their parent nodes.

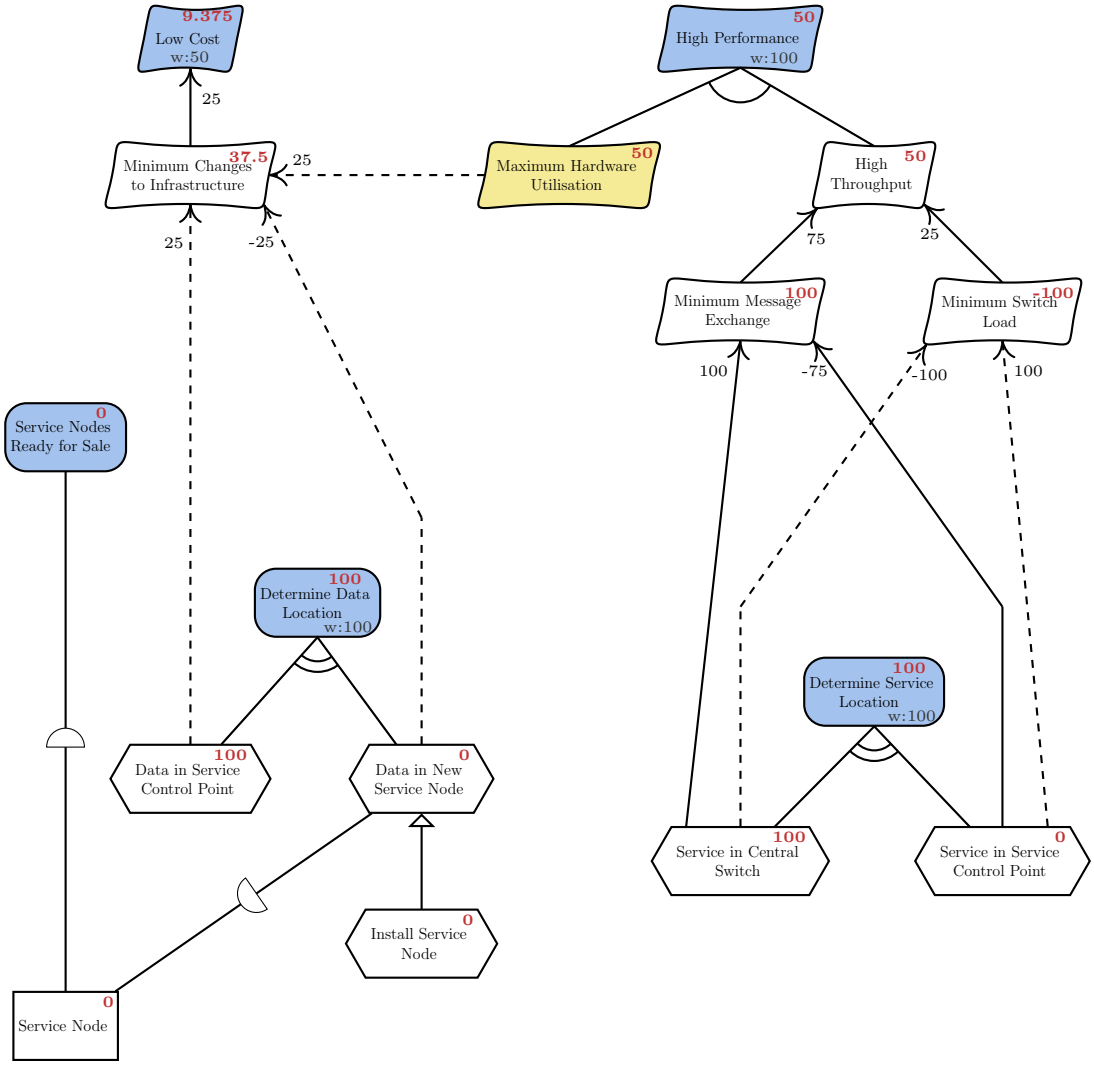


Figure 4.8: Results of the optimisation schema utilising initialisation values as applied to the Telecommunication Exemplar.

Therefore, the results of a capped value or tolerance approach can also been seen in figure 4.8, assuming initialisation values are used. If initialisation values are not used then the the results are the same as those presented in figure 4.7.

4.4.2 NetME Interface

The test cases presented both in this subsection and the next were developed in conjunction with a group of software engineering students, undertaking their final year project at Curtin University. Their project was to develop a tool that could be used to create GORE based graphs and eventually implement existing evaluation strategies for these graphs.

The group decided their first task was to develop a highly customisable modelling environment, referred to as *NetME*. The students defined NetME as below:

> *"NetME is designed as a highly extensible and abstract modelling language environment designed for the implementation of arbitrary modelling languages and frameworks with as few restrictions imposed on the developer as is possible." (Team Leader)*

Figure 4.9 presents the GRL graph that was developed in conjunction with the project group in order to model the goals, tasks, and resources needed to complete NetME. Once again, table 4.28 presents the size breakdown of this graph; the graph is both larger and more complex than the previous Telecommunication Exemplar, having almost double the number of nodes and relationships. However, it must be noted that due to inexperience only positive contributions were modelled.

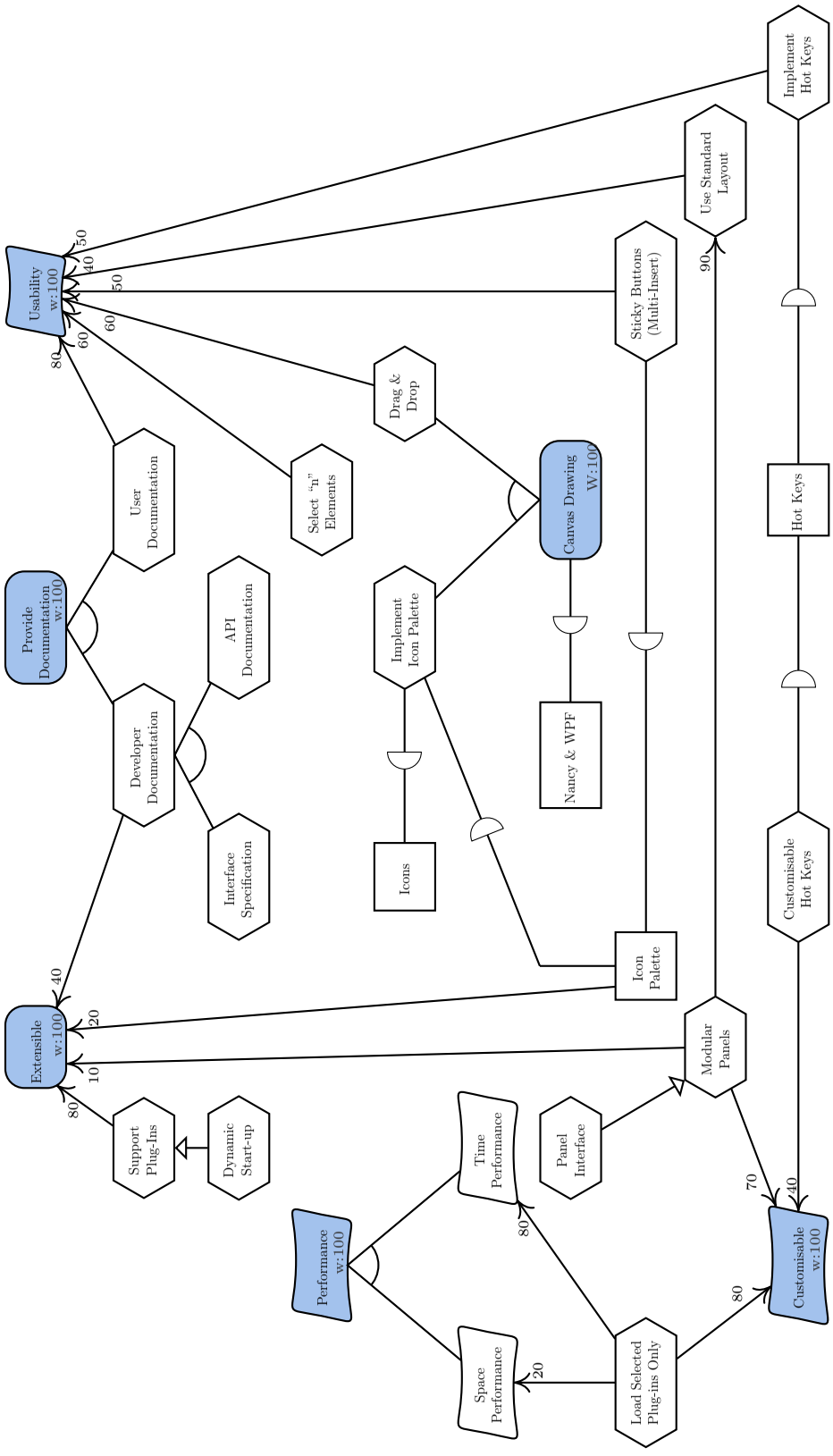


Figure 4.9: GRL Graph modelling implementation of the NetME interface.

Nodes	**29**
Goal	3
Softgoal	6
Task	16
Resource	4
Relationships	**29**
Decomposition	6
Correlation/Contribution	16
Dependency	6

Table 4.28: Size breakdown of the NetME GRL Graph.

4.4.2.1 Basic Optimisation Schema

Figure 4.10 shows the results of the basic optimisation schema as applied to the NetME Interface; the final objective value for this graph was 520 out of a maximum 600. This test case presents a unique example of the limitations of the optimisation schema, as no negative contributions were modelled there were no trade-offs to be made. Given the structure of the graph, all root nodes achieved their highest possible score, and the only point at which tasks did not was if their parent nodes were already at the maximum and hence, their values would have no effect. The goal of optimisation is to find the best solution given a range of alternatives, situations such as this present no alternatives.

4.4.2.2 Optimisation Schema with Initialisation

In this test case, the basic optimisation offered little enlightenment to the situation. However, after analysing the results and considering external factors — such as time constraints — the team decided they disliked the number of tasks receiving a maximum score. Initialisation values were provided to several of the tasks and the optimisation process repeated, the nodes provided with initialisation values have again been highlighted on the graph. Figure 4.11 shows the results of the optimisation process with these initialisation values; the final value of the objective function this time was 470. However, the results still largely suffer from the same issues as

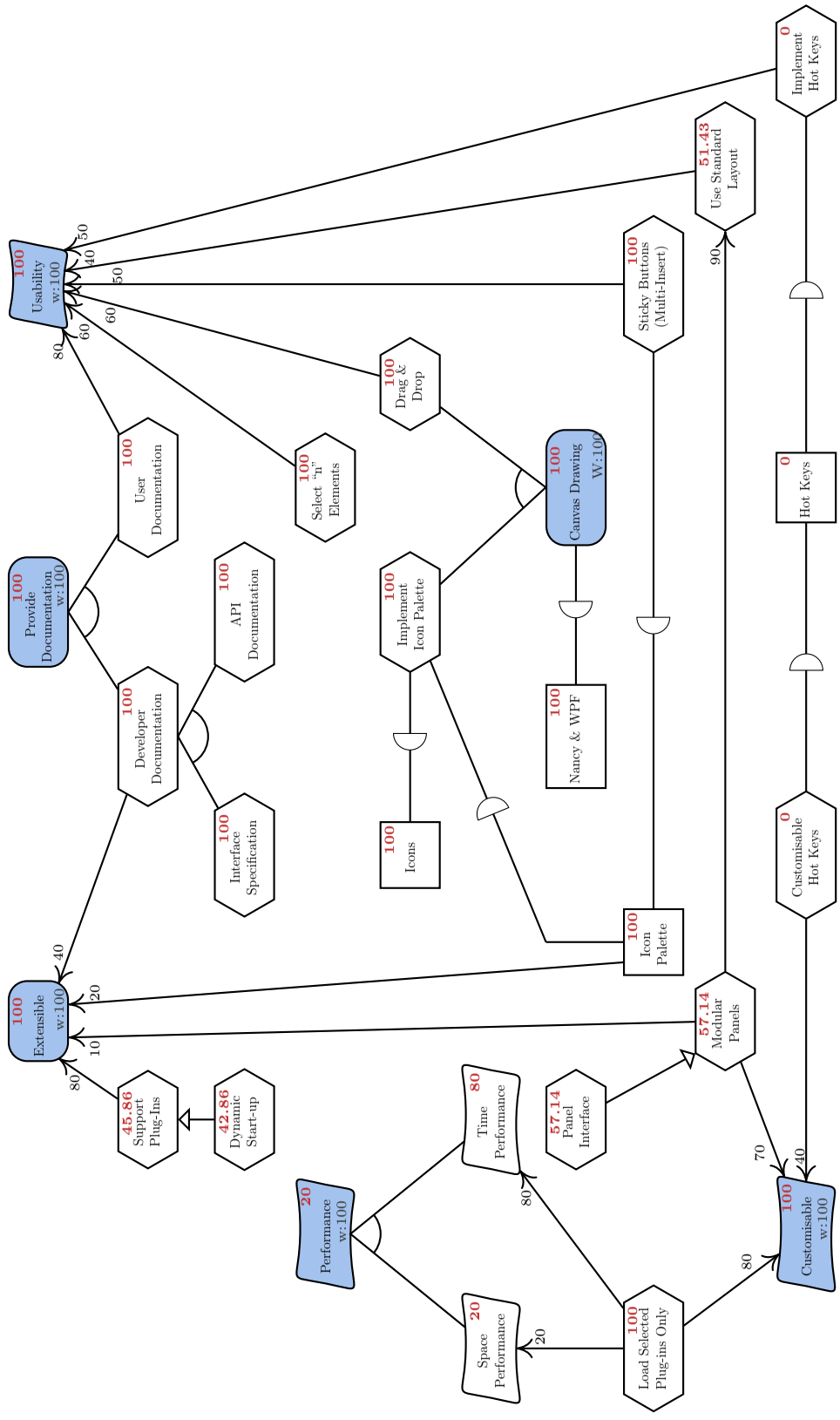


Figure 4.10: Results of the basic optimisation schema as applied to the NetME Interface.

before.

4.4.2.3 Optimisation Schema with Tolerance Evaluation

While the NetME Interface contains no negative contributions, its highest contribution score is 90, yet most of the root nodes have received a score of 100. To address this, the optimisation was run again, applying *both* initialisation values and a tolerance. Figure 4.12 shows the results of the optimisation when a tolerance is taken into consideration; the final value of the objective function was 440. At this point none of the root nodes have achieved their maximum score, even though a majority of tasks do. Given these results it is clear to see that the graph itself may require refinement.

4.4.2.4 Optimisation Schema with Capped Values

As the NetME Interface contained no negative contributions, it was impossible for any node to receive a negative value. Therefore, application of capped values would be redundant as the results would be based entirely on the other evaluation options.

4.4.3 NetME Plugin

NetME is a *modelling environment*, it relies on plugins to provide specific language or framework capabilities. Hence, the group's second task was to develop a GRL plugin that allowed for the creation and evaluation of GRL graphs. Figure 4.13 shows the GRL graph — developed in conjunction with the group — for the plugin.

Table 4.29 outlines the size of the graph used to model the GRL plugin; the size and complexity of this graph are significantly higher than either the Telecommunication Exemplar or the NetME Interface. In contrast to the previous example,

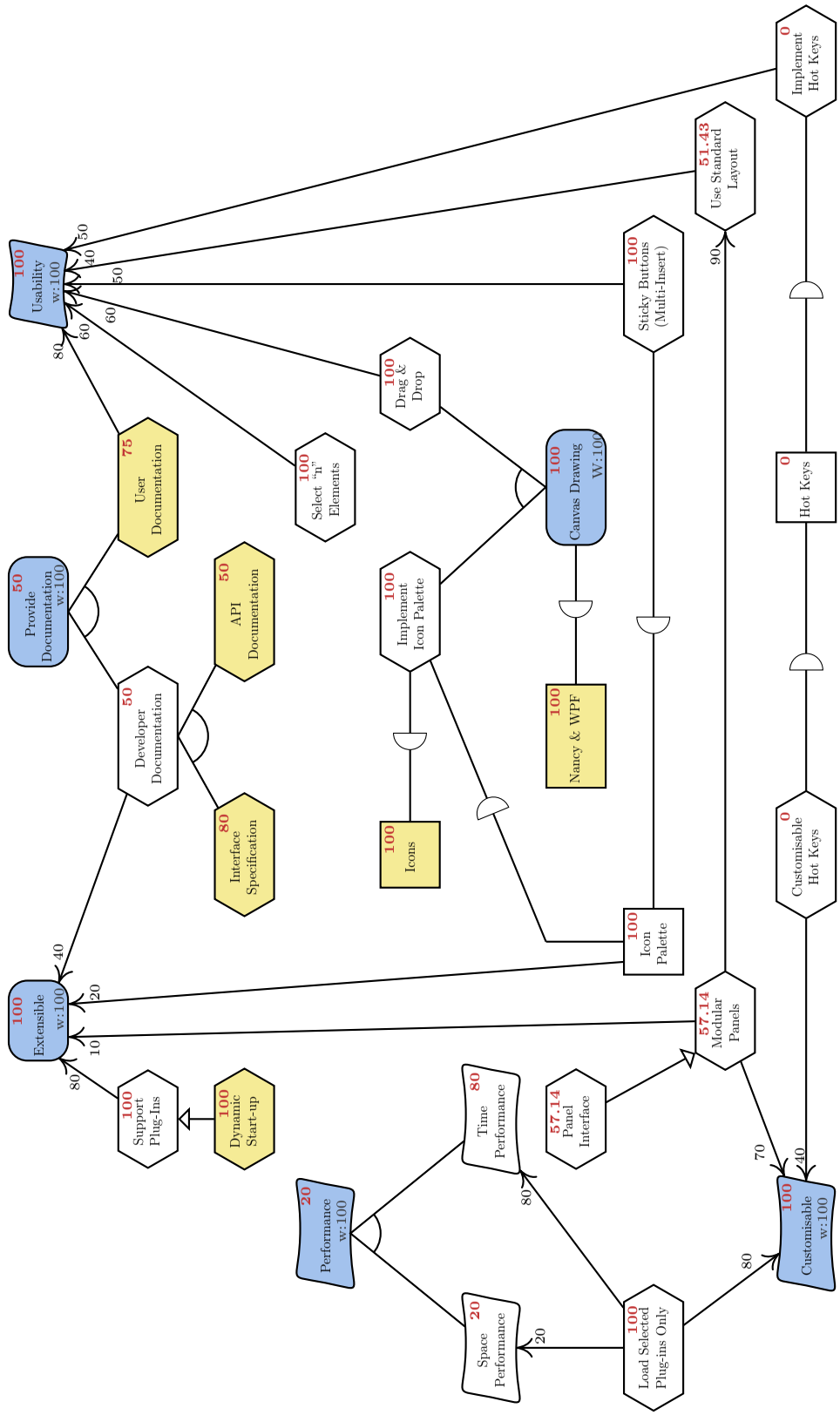


Figure 4.11: Results of the optimisation schema as applied to the NetME Interface with use of initialisation values.

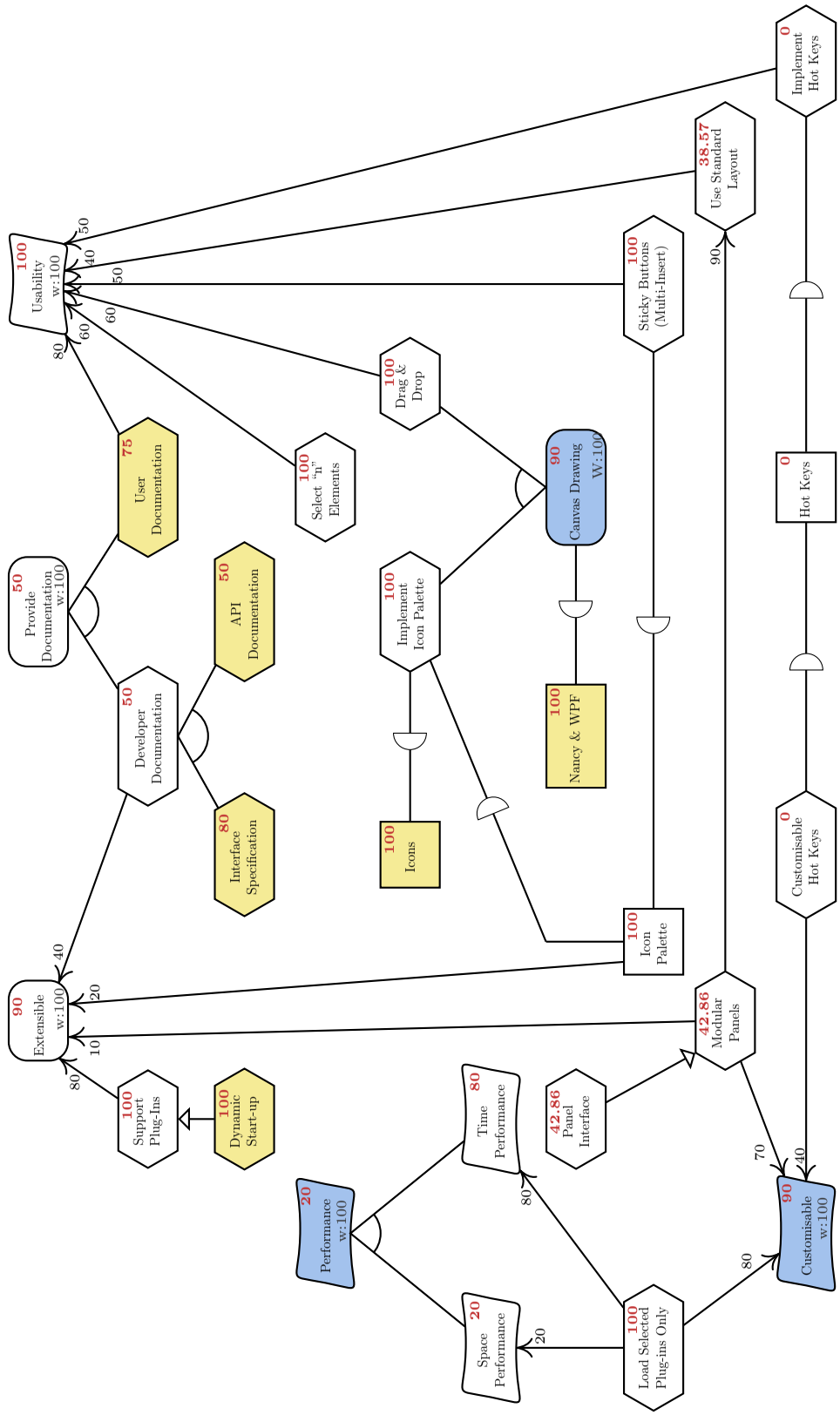


Figure 4.12: Results of the optimisation schema as applied to the NetME Interface with use of initialisation values.

Nodes	**41**
Goal	3
Softgoal	4
Task	29
Resource	5
Relationships	**39**
Decomposition	17
Correlation/Contribution	16
Dependency	6

Table 4.29: Size breakdown of the GRL Plugin Graph.

this graph models both positive and negative relationships, creating conflict within the graph, and hence, an opportunity for optimisation to effectively choose the best scenario.

4.4.3.1 Basic Optimisation Schema

Figure 4.14 shows the results of the basic optimisation schema as applied to figure 4.13; given this evaluation strategy the final value of the objective function was 349.01 of a possible 400. The only node that failed to achieve its best value was *Performance*, upon examination it is clear that the reason for this shortfall is that one of the decomposed children — *Space Performance* — can, at most, provide a score of 60. Due to the benefit towards *Usability*, *Check On Insert* prevents the full value of 60 being applied.

4.4.3.2 Optimisation Schema with Initialisation Values

Once again, the development team decided that some of the node values were unrealistic, such as *User Documentation*. Additionally, some tasks were deemed either unappealing or too complex to implement within the time limits, such as *Select Visible Value* and *Display During Execution*. To implement their preference, the optimisation schema was evaluated again, this time with use of initialisation values.

Figure 4.15 shows the result of these initialisation values, again the nodes assigned initialisation values are highlighted; resources associated with external actors were also provided with initialisation values to ensure their values were not changed. In this evaluation, the objective function received a final value of 310. While the objective score may be lower than before, it is more realistic. Additionally, due to the effect of the initialisation, *Check On Insert* now receives a zero value, hence *Performance* can achieve its best possible score of 60.

4.4.3.3 Optimisation Schema with Initialisation and Tolerance

Upon examination of *Usability* and *Time Performance*, it became apparent that the multitude of relationships were resulting in the node achieving a maximum score even though no individual relationship is strong enough to warrant such a value. This issue can be addressed by using a tolerance evaluation as shown in figure 4.16, the final value of the objective function is now 300. This is a small improvement, though the change to *Time Performance* does not propagate to *Performance* as its sibling *Space Performance* already imposes a stricter restriction.

4.4.3.4 Optimisation Schema with Initialisation, Tolerance, and Capped Values

After applying the initialisation values, one major affect was overlooked, several nodes with negative impacts now have negative values. After consideration, the development team decided they only wanted to model their actual progress towards goal achievement. Use of the capped values during the optimisation evaluation ensures this goal can be achieved.

However, when evaluating the model with the addition of capped values, the calculation time went from 18 seconds to over 24 hours. Figure 4.17 shows the results that were eventually obtained; while the complexity of the model was greatly increased, the eventual production of results proves that the model remained feasible. Inclusion

of capped values has several key effects on the results, most notably *Time Performance* has greatly decreased as the denial of *Display During Execution* no longer contributes towards its score. *Usability* also has several changes, the denial of *Via Pop-Up* no longer positively affects the score, so the value of *On-Insert Warning* has been increased in order to maintain the highest possible *Usability* score.

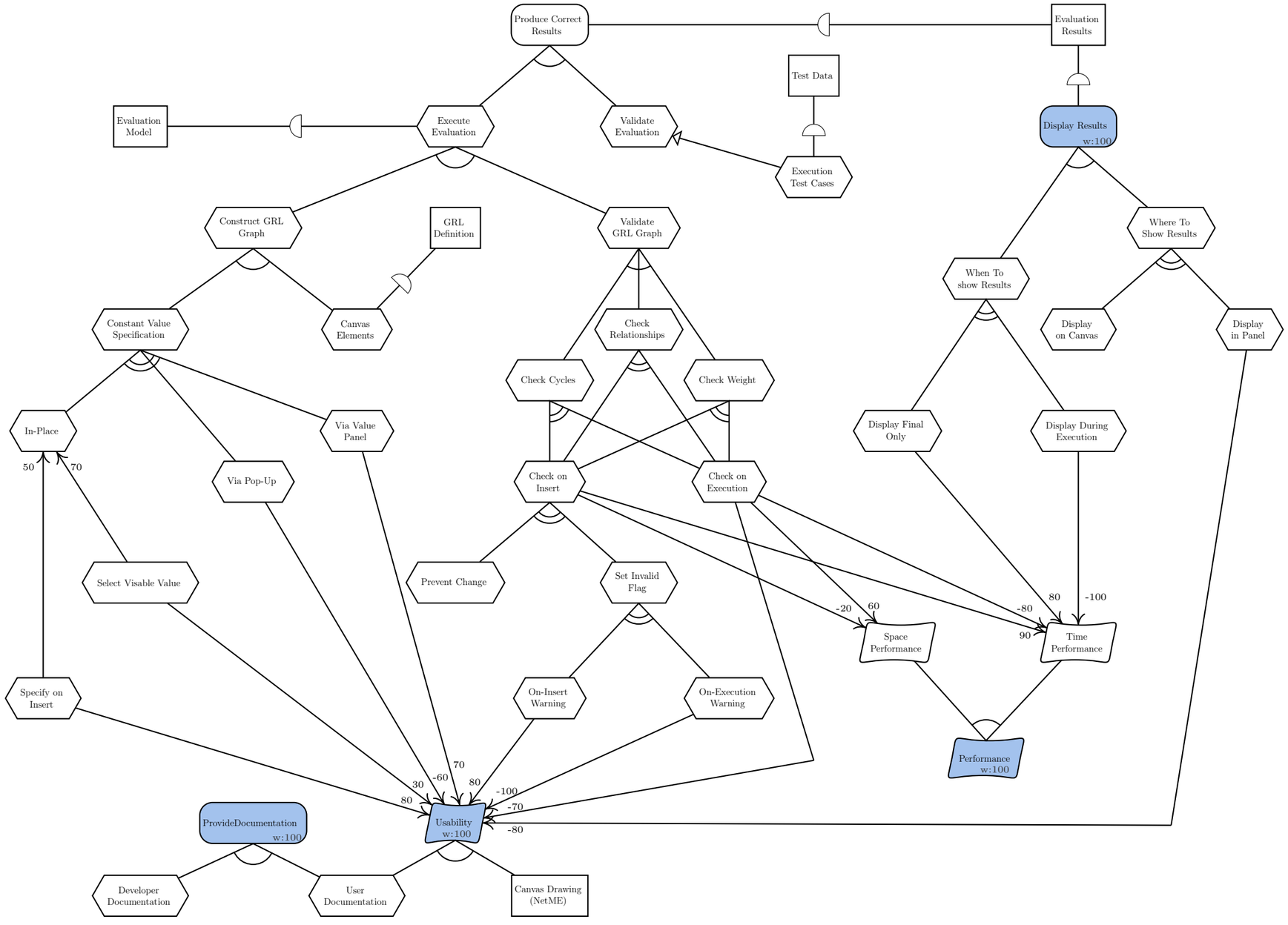


Figure 4.13: GRL Graph modelling the implementation of a plugin capable of GRL evaluation for NetME.

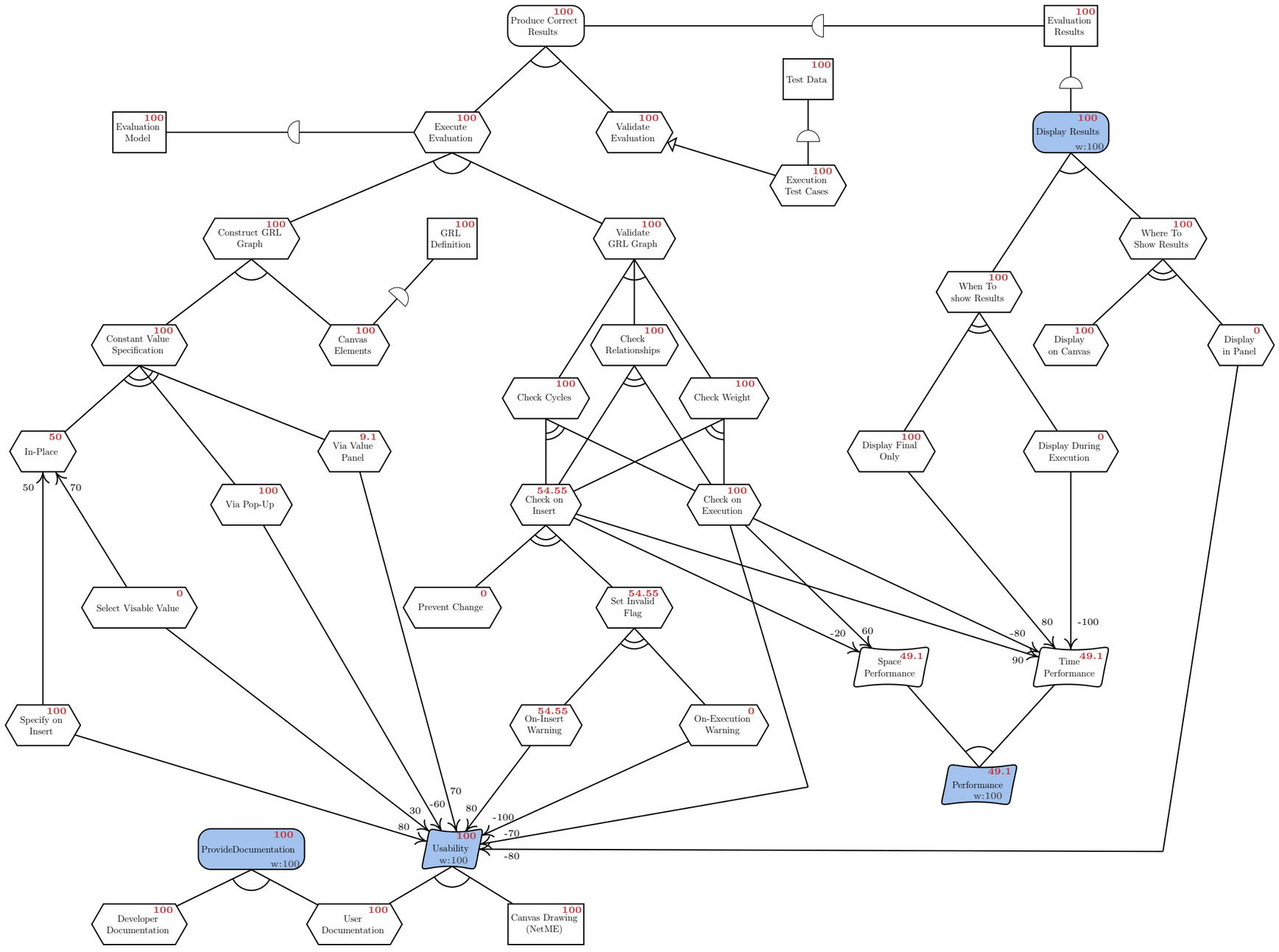


Figure 4.14: Results of the basic optimisation schema as applied to the NetME GRL Plugin.

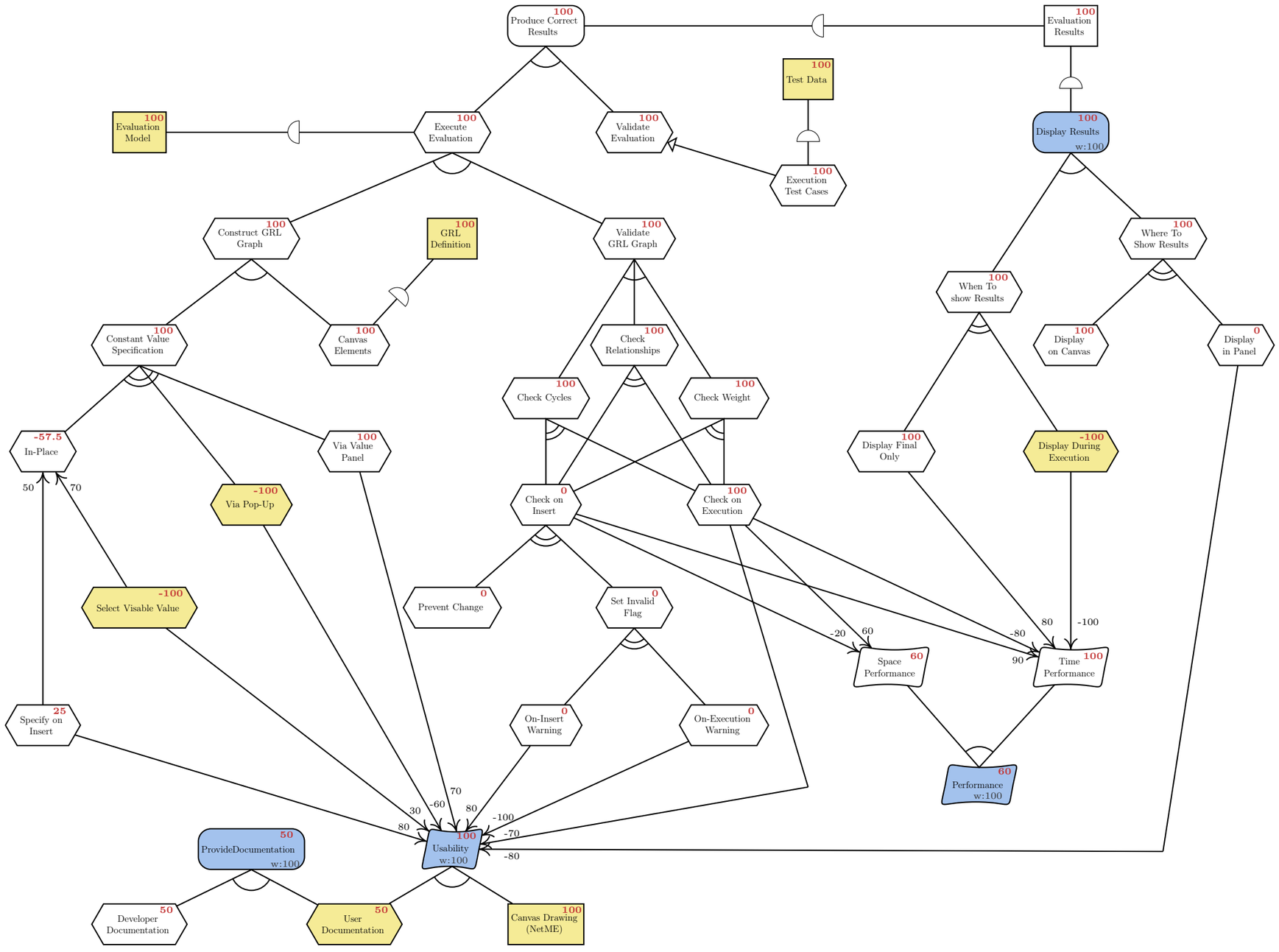


Figure 4.15: Results of the optimisation schema as applied to the NetME GRL Plugin with use of initialisation values.

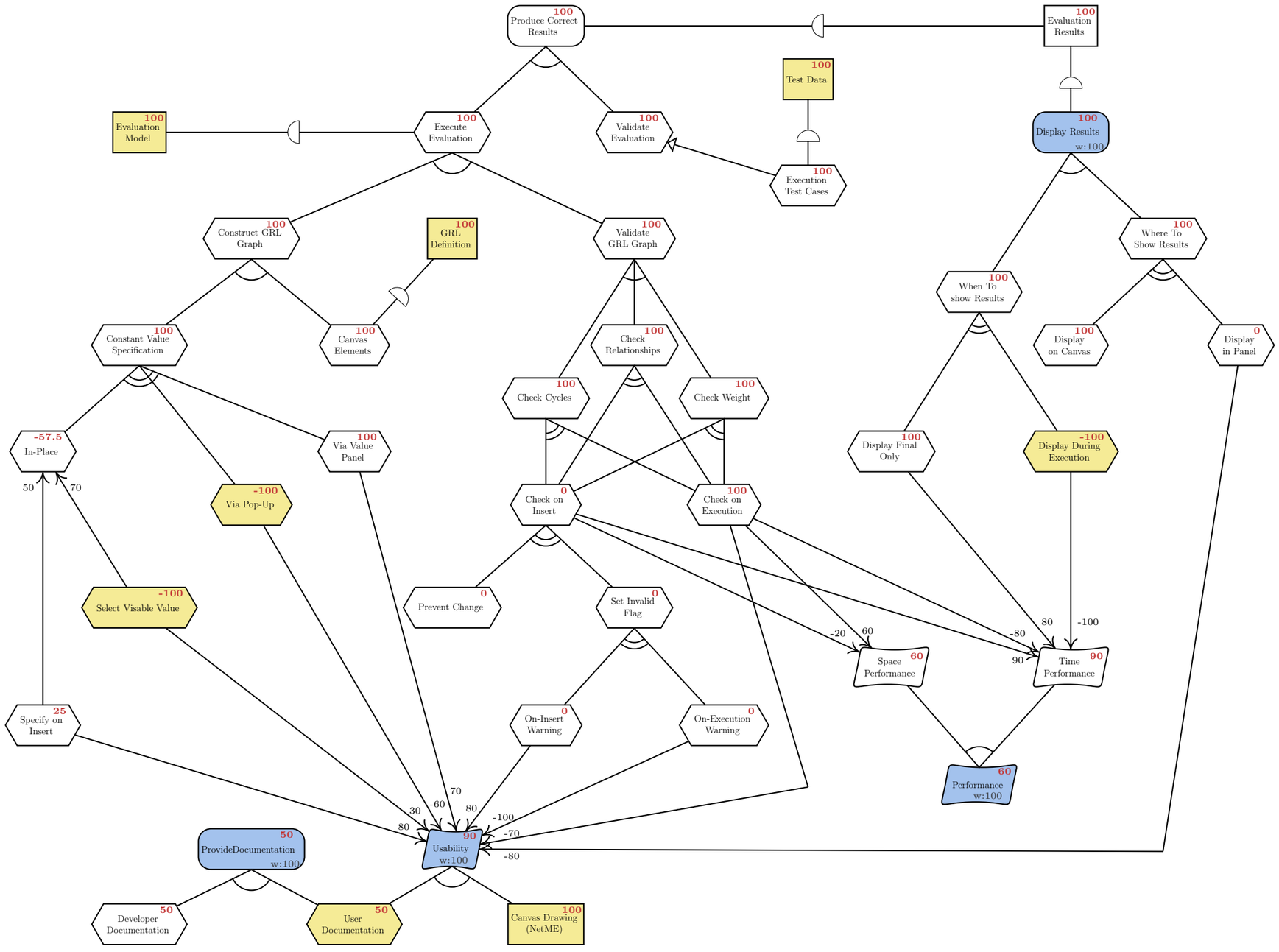


Figure 4.16: Results of the optimisation schema as applied to the NetME GRL Plugin with use of initialisation values and tolerance evaluation.

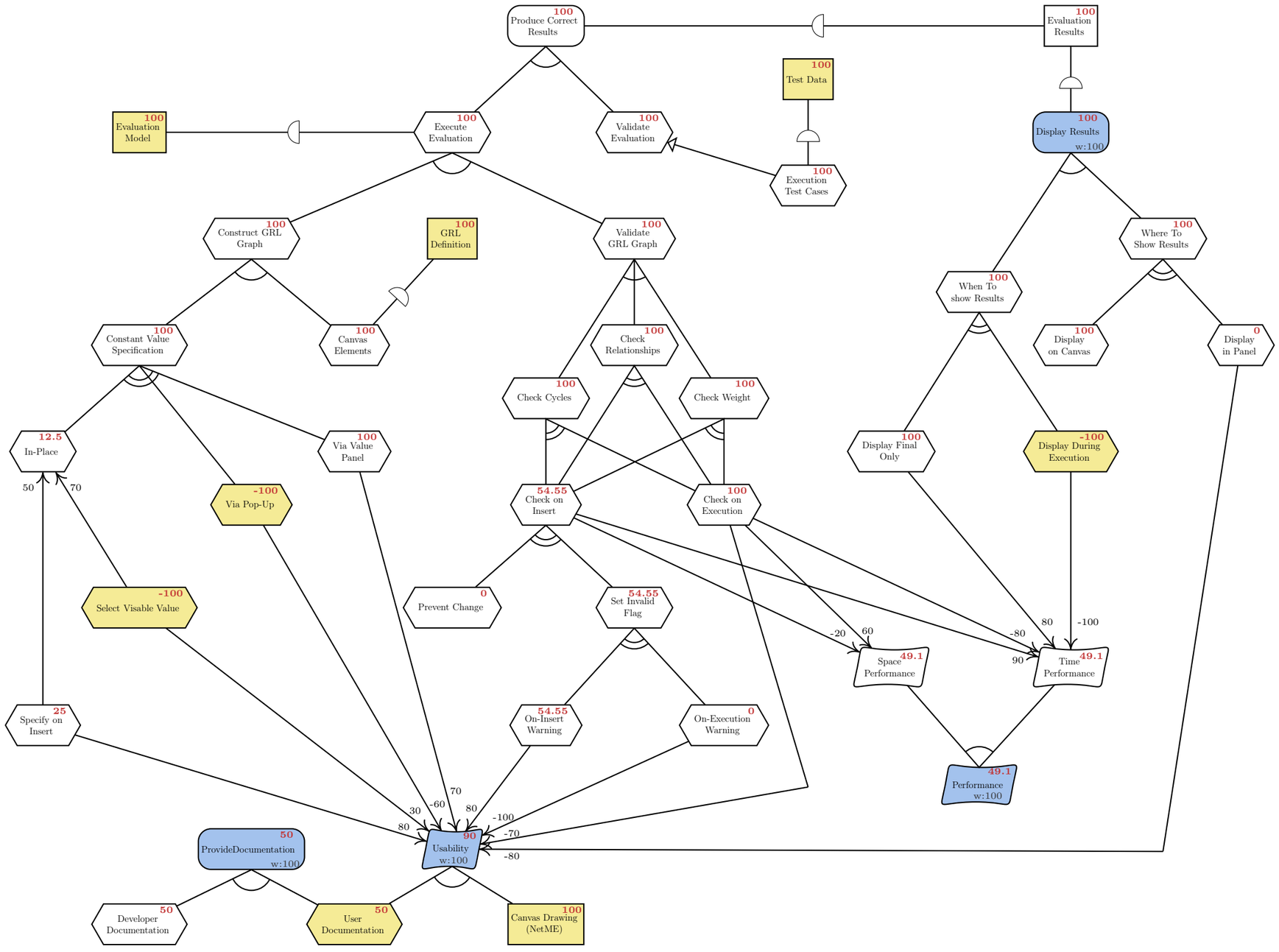


Figure 4.17: Results of the optimisation schema as applied to the NetME GRL Plugin with use of initialisation, capped values and tolerance evaluation.

Evaluation Option	Paper Graph	NetME Interface	NetME Plugin
Default	$0.016s$	$0.015s$	$0.061s$
Initialisation	$0.031s$	$0.016s$	$0.062s$
Capped Values	$2.62s$	$0.032s$	$> 24hrs$
Tolerance	$0.016s$	$0.047s$	$> 24hrs$
Initialisation & Capped Values	$< 0.01s$	$0.031s$	$> 24hrss$
Initialisation & Tolerance	$0.047s$	$0.031s$	$0.187s$
Capped Values & Tolerance	$2.933s$	$0.063s$	$> 24hrss$
All	$0.015s$	$0.031s$	$> 24hrs$

Table 4.30: Time comparisons of all evaluation options for the GRL optimisation schema, across the three test cases.

4.4.4 Time Analysis

Given the three evaluation options provided by the optimisation schema, there are eight possible evaluation combinations available. The previous sections did not show all eight combinations for every test case, rather, specific options were chosen based on the structure of the individual graphs. As discussed above, this omission was made as the results of some combinations were often identical. Table 4.30 shows the computation time for all eight evaluation options, as applied to all three test cases[3]. As clearly shown, the first two test cases always produced results in under three seconds, however, as addressed in section 4.4.3, the third test is considerably larger, and when the capped value option is applied the computation time drastically increases.

[3]The specified times were determined on the same machine as outlined in section 3.6.2; 64-bit 3.30GHz Windows 7 desktop with 8GB RAM.

4.5 Schema Restrictions and Drawbacks

Previously it was established that, as a language, GRL can be used to model an unlimited number of situations. While the presented optimisation schema maintains a large portion of this polymorphism, there are a handful of situations that can create an infeasible model or skewed results. A model becomes infeasible when a solution cannot be found that satisfies all constraints. Alternatively, a model could be created that produces an untrustworthy result.

This section covers the limited situations in which the schema either breaks down, or struggles to generate trustworthy results in a timely manner. It is important to note that the two situations which create infeasible models will never occur in a well constructed graph. If they are constructed due to oversight or inexperience, then any IDE or simulation would be able to detect the occurrences and generate a warning. The remaining two situations which may impact the trustworthiness of the results are unlikely to occur under traditional uses of the language, again warning messages could be produced should they occur unintentionally.

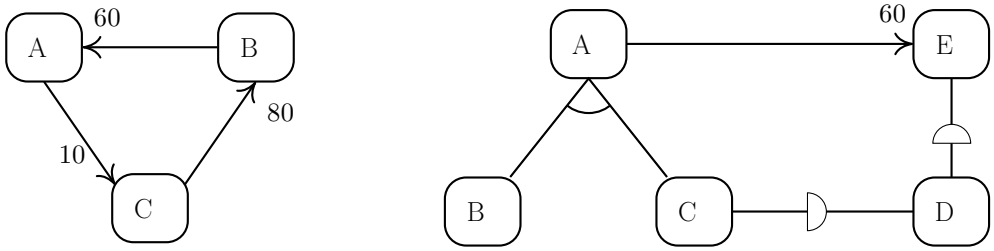
4.5.1 Cycles

A cycle in the graph exists if you can start at a node and, following only relationships to parent nodes, find your way back to the original node, as shown in figure 4.18. Cycles should not exist in most models, as they would indicate a presuppose dilemma which cannot be resolved. For example, the graph in figure 4.18a is a simplistic cycle, equation (4.22) shows a simplified version of the accompanying constraints. The only way these constraints can be satisfied is by assigning all nodes a value of zero.

$$A = 0.6 * B \tag{4.22a}$$

$$B = 0.8 * C \tag{4.22b}$$

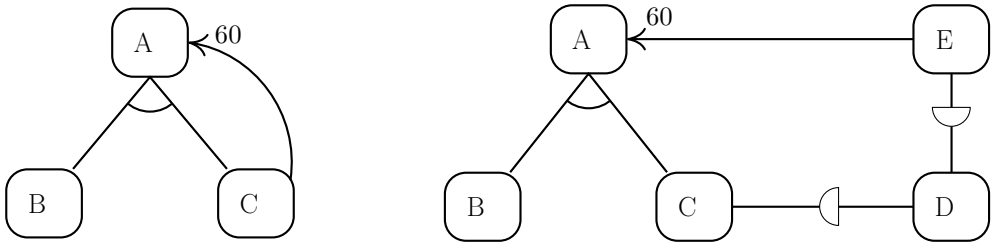
$$C = 0.1 * A \tag{4.22c}$$



(a) A simple 3 node graph

(b) A complex 5 node graph

Figure 4.18: Possible methods of generating cycles in a GRL graph, in a simple manner (A), and a more complex manner (B)



(a) A simple 3 node graph

(b) A complex 5 node graph

Figure 4.19: Possible methods of a single node contributing twice to it's parent, in a simplistic, unrealistic manner (A), and a more complex manner (B)

Figure 4.18b shows a more complex example of how a cycle can be formed. If a graph is constructed by hand then as it increases in size, it is feasible for a cycle to accidentally be formed.

4.5.2 Multiple Effects

This situation occurs when a single node is linked to a parent by more than one relationship, as shown in figure 4.19. Similar to the previous situation, this should not arise unless there is a flaw in the graph. Figure 4.19b illustrates a situation where node C has a direct affect on node A through a decomposition relationship, as well as an indirect affect though nodes D and E. In large graphs it is feasible that a situation such as this could be accidentally modelled.

4.5.3 Root Node without Importance

A root node without an importance does not create an infeasible model, rather, it will greatly increase required computation time. This increase occurs as the optimisation solver must completely explore every possible solution involving the node in question. When all root nodes have a weight, solvers typically use a branch and bound approach to determine which solutions are invalid. With a weight of zero, any possible value will also have a zero impact on the objective function, the solver cannot use deductive reasoning to realise this so it must mathematically test every value.

4.5.4 Non-root Node Importance

If a non-root node is assigned a non-zero importance, then it effectively contributes to the objective function in two ways. As opposed to the other issues, this does not create an infeasible model or result in largely increased computation times. However, any results generated in this circumstance will be largely skewed towards the child node.

4.5.5 Graph Quality

The main issue with applying optimisation to a GRL Graph is the underlying quality of the graph. The issues presented above can easily be verified in order to prevent an infeasible optimisation model. However, as section 4.4.2 demonstrated, if the quality of the graph is lacking, either due to incompleteness — no negative relationships — or a lack of experience, then the results of the optimisation are negligible.

4.6 Lessons Learned

Applying linear programming to a broad situation, such as the goal-orientated requirements language, brought to light several issues that were not discovered in chapter 3. This section briefly touches on the main issues that were encountered in developing the GRL optimisation schema.

4.6.1 Applying a Non-Binary Unknown Limit

The NFR optimisation schema used a binary variable to determine the minimum or maximum of another binary variable, this process relied heavily on the definition of the variables as *binary*. The GRL optimisation schema, however, required this process be applied to continuous variables. That is, a variable was required to take the maximum or minimum value of a set of other variables, this goal was achieved by combining the limit calculation with the previous binary approach. Consequently, a series of variables had to be defined for each parent-child *relationship*, rather than just for each node.

4.6.2 Pseudo-Linear Evaluation

The NFR optimisation schema does require that the value of one node carry over to another, however it does not require the gradual addition of relationships to the *same* node. Section 2.6 elucidated the non-linear nature of mathematical optimisation, in order to overcome this and simulate the linear evolution of a node's score, multiple variables needed to be utilised. By assigning each "step", to its own variable and then using that variable in the calculation of the next "step" a linear progression could be simulated.

4.6.3 Importance of Graph Structure

As iterated earlier, GRL is a language with very few limitations, while the developed schema was flexible enough to maintain this ideology it became apparent that just because any graph structure *can* be created does not mean it *should*. While usually these graph taboos would just be indicative of poor judgement or a lack of experience, they do have the unfortunate side effect of greatly increasing the computation time of the optimisation schema. However, careful consideration of these issues does reveal they could be easily detected by an algorithm.

4.7 Summary

This chapter presented a dynamic, customisable optimisation schema for the goal-oriented requirements language. Different evaluation strategies allow for the use of initialisation values to influence results, capped values to prevent negative multiplication, and use of a tolerance so goals cannot be completely achieved by the existence of small contributions alone. Section 4.3 demonstrated the various constraints and how they enforce the desired conditions. After verification of the schema, section 4.4 applied various combinations of evaluation strategies to a noted exemplar, and two real world graphs of increasing complexity, comparing both results and computation time. Section 4.5 addressed the restrictions of the optimisation schema, providing both details and examples on each situation that could potentially hinder the optimisation process. Finally, section 4.6 detailed the major lessons that were learned from applying mathematical optimisation to the goal-orientated requirements language.

The schema presented in this chapter is based on the habitual practise of modelling the degree to which a node was both satisfied *and* denied. While this approach is historically common, it does present fundamental flaws when designing a quantitative evaluation method aimed at assisting development decisions. In order to provide accurate, useful feedback to developers, an evaluation option was included

in the optimisation schema that forced it to consider only evidence of a nodes satisfaction and consequently an indication of its acceptance or rejection in the final system. Unfortunately, when applied to graphs of a significant size, the complexity of the generated optimisation model caused significant time delays.

The next chapter develops an alternative optimisation schema that is based on the idea of only modelling evidence of a nodes existence; incorporating this approach in the initial schema removes the necessity of defining additional constraints. Hence, the complexity of the generated models would not be affected and the computation time would not increase.

Chapter 5

Goal Requirements Language — Effect Based Alternative

The previous chapter developed and implemented an optimisation schema for the goal-orientated requirements language, the developed optimisation schema presented several alternative evaluation strategies that can be applied either individually or in combination. By default, the presented optimisation schema subscribed to the idea that denying something bad has a measurable positive effect; one of the evaluation options for the schema subverted this paradigm by only modelling evidence that directly contributed towards a nodes achievement, this evaluation option was refereed to as *using capped values*. Unfortunately, this evaluation method greatly increased the evaluation time of the optimisation schema when applied to graphs of a considerable size.

In a quantitative evaluation model, used to assist in design decisions and implementation selections, this evaluation option is of vital importance as not implementing something would not actually result in a goal being achieved. In order to overcome the scalability issues of the previous optimisation schema, this chapter presents an alternative optimisation schema that only models evidence of a node's existence. By incorporating this idea from the beginning, additional constraints are no longer required, resulting in models of significantly less size being constructed.

This chapter presents an *alternative* approach to the goal-orientated requirement language, as such the language structure provided in section 4.1 is still applicable. Section 5.1 illustrates the necessary changes to the previous optimisation schema and then presents the complete alternative optimisation schema. Section 5.2 applies the alternative optimisation schema to the same test cases presented in section 4.4

comparing both the results and the evaluation time. Finally, section 5.3 will explain the lessons learned from implementing the alternative optimisation schema, focusing on the relationship between constraints and computation time.

5.1 Optimisation Schema

In order to overcome the afore-mentioned time complexity issues, the optimisation schema was modified to only model the effects of a node's existence. Figure 5.1 outlines the necessary modifications to the original schema, constraints marked for removal have been crossed out, and those requiring modification have been highlighted; figure 5.2 presents the final flow graph for the alternative optimisation schema utilising the equations presented in this section. In addition to these constraint modifications, the majority of variable bounds also need to be changed. This reflects the major difference between the two optimisation schemas, the previous schema modelled how badly a node was effected by allowing it to take a negative score, in this version, nodes only model how close they are to being achieved, that is, they can only take positive scores.

Demonstration Graph: The demonstration graph from the previous chapter shall also be utilised to demonstrate this optimisation schema, the final optimisation models can then be compared to elucidate the differences. The demonstration graph only requires one modification; *Resource1* cannot have a negative initialisation value as the schema only models the degree to which a node is achieved. This modification is shown in figure 5.3.

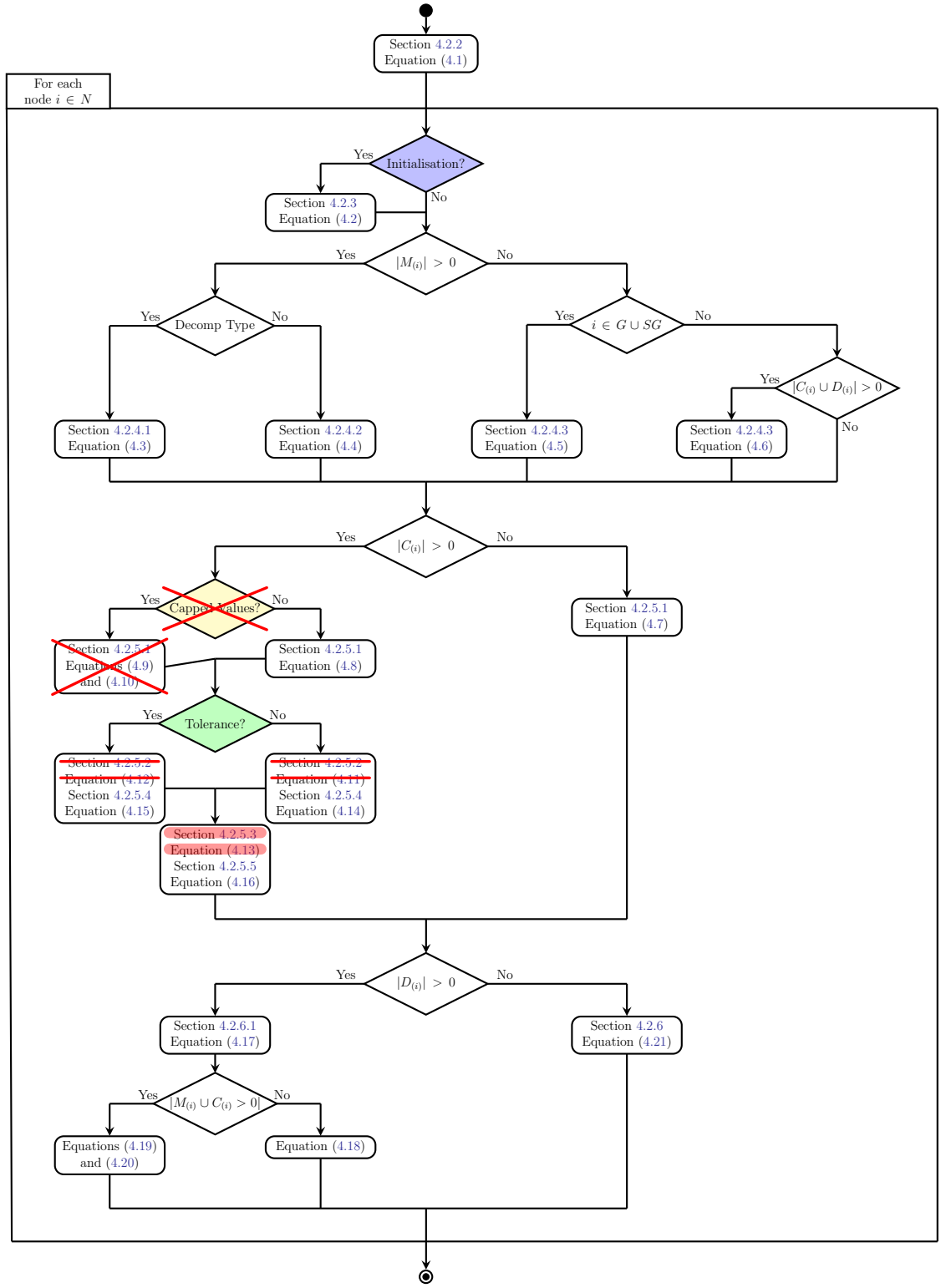


Figure 5.1: The flowchart of the original optimisation schema — figure 4.1 — showing the modifications necessary for the alternative schema.

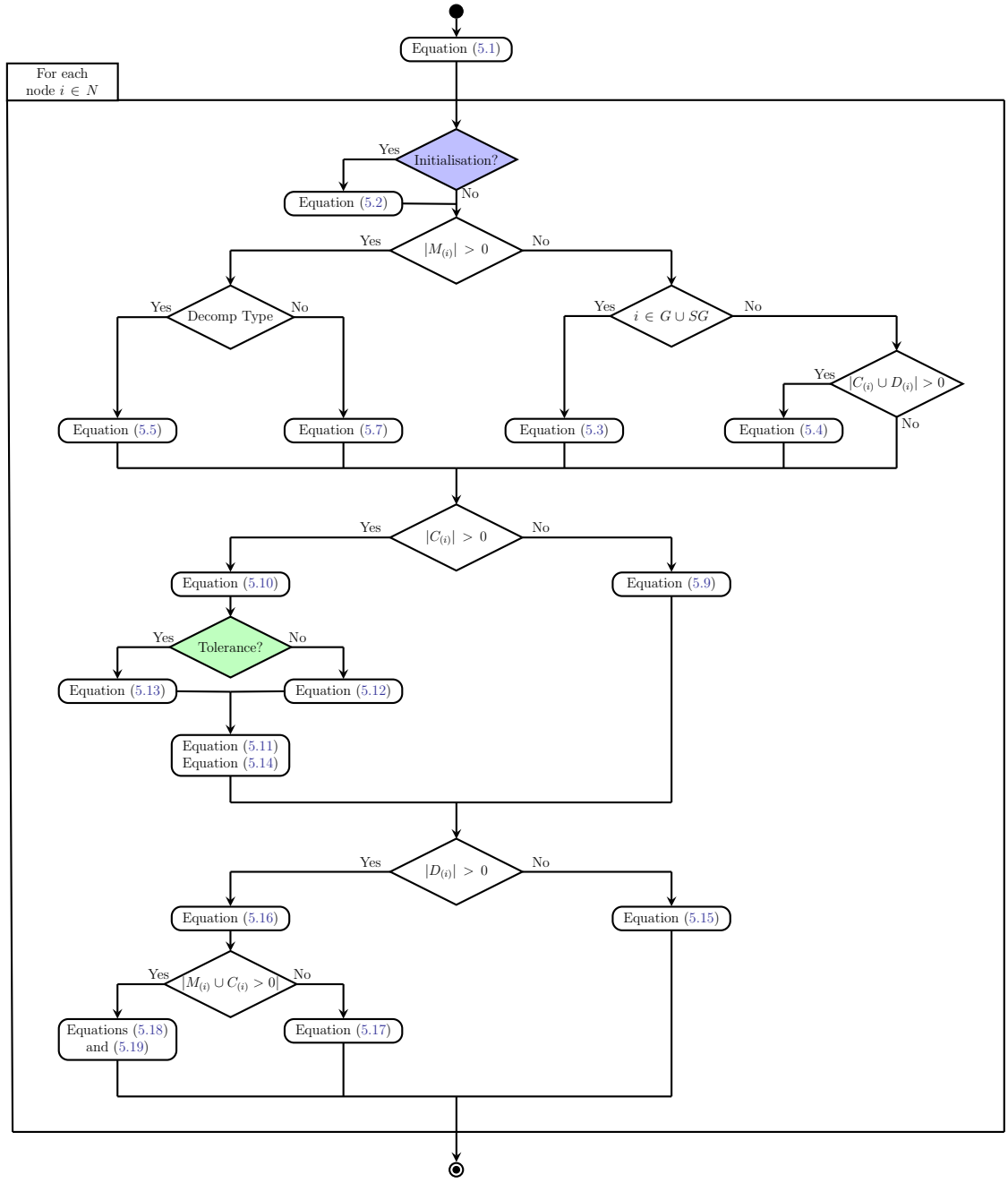


Figure 5.2: The flowchart of the original optimisation schema, showing necessary modification

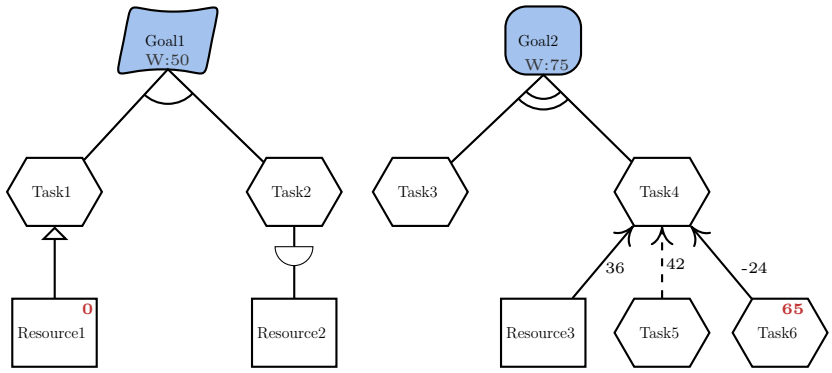


Figure 5.3: The demonstration graph from figure 4.2 with initialisation values modified for use with the alternative optimisation schema.

5.1.1 Graph Notation, Objective Function, and Node Initialisation

Table 5.1 reiterates the unchanged graph notation from the previous chapter. Table 5.2 presents the objective function and initialisation equations. While the equations themselves are unmodified, the lower bounds of the associated variables have been changed to reflect the non-negative restriction of this alternative optimisation schema.

Demonstration Graph Table 5.3 shows the constraints added to the optimisation model for the objective function and node initialisation based on the alternative schema. These constraints are largely unchanged from the original optimisation schema. The one change mirrors that of the demonstration graph itself, *Resource1* is initialised to 0 to show that the resource shall not be available or implemented.

5.1.2 Decomposition and Means-End Relationships

Table 5.4 presents the constraints and variable bounds needed to model decomposition relationships. As these equations are not affected by the modified value range, they are unchanged from the original optimisation schema.

Notation	Explanation	Remarks
$N = G \cup SG \cup T \cup R$	The sets G, SG, T, R respectively represent the goals, softgoals, tasks and resources.	The generic notation N is used if the node type is irrelevant. Nodes are typically detonated by i, j etc ...
$E = decomp \cup contrib \cup dep$	The set of directed arcs, *decomp* — relationships defined as Decomposition or Means-End *contrib* — relationships defined as Contribution or Correlation *dep* — relationships defined as Dependency	For an arc (i, j) between nodes i and j, node i is considered the parent and node j is considered the child.
$M_{(i)}$	The set of nodes related to node i by decomposition or means-end relationships. Where node i is the parent.	Constant
$C_{(i)}$	The set of nodes related to node i by contribution relationships. Where node i is the parent.	Constant
$D_{(i)}$	The set of nodes related to node i by dependency relationships. Where node i is the parent.	Constant

Table 5.1: The defined graph structure used in GRL models.

Section	**Variables**	**Constraints**
Objective Function	$(0 \leq dVal_{(i)} \leq 100)$ $(0 \leq imp_{(i)} \leq 100)$	$\max \sum_{i \in N} \left(\frac{imp_{(i)}}{100} \cdot dVal_{(i)} \right)$ (5.1)
Initialisation	$(0 \leq default_{(i)} \leq 100)$ $(0 \leq mVal_{(i)} \leq 100)$	Given $\lvert M_{(i)} \rvert + \lvert C_{(i)} \rvert + \lvert D_{(i)} \rvert = 0$ and $default_{(i)} \neq NaN$ define: $mVal_{(i)} = default_{(i)}$ (5.2)

Table 5.2: Constraints and variable bounds required for the initial steps of the alternative optimisation schema

Options	Equation	Generated Constraints
All	(5.1)	$\max\left((0.5 \times dVal_{Goal1}) + (0.75 \times dVal_{Goal2})\right)$
Init	(5.2)	$mVal_{Resource1} = 0$
Init	(5.2)	$mVal_{Task6} = 65$

Table 5.3: The objective function and initialisation constraints generated for the *demonstration graph* presented in figure 5.3.

Demonstration Graph Table 5.5 presents the constraints generated to implement decomposition in the *demonstration graph*. based on table 5.4.

5.1.3 Contribution and Correlation Relationships

As demonstrated in section 4.2.5 and again in table 5.6, the process of modelling contribution and correlation relationships can be divided into several steps. While the bounds of all associated variables need adjusting in this new schema, only a select few of the constraints need to be modified. The modifications made to each of these steps are as follows:

No Contribution Children: Patently, there are no changes necessary when no contribution relationships exist.

Basic Contribution Score: This equation is the companion of equation (4.8), in this version of the schema no alternative capped version is required, as $dVal_{(j)}$ is already within the range $(0, 100)$.

Apply Lower Limit: As a node's score is now capped at zero, there is no longer any need for a negative tolerance, hence the associated constraints are removed and the lower limit is fixed at zero.

Determine Tolerance and Apply Upper Limit: Use of capped values has no effect on positive node scores so these constraints remain unchanged.

Demonstration Graph The constraints generated for the *demonstration graph* to enforce contribution and correlation relationships is where the two models differ

Decomposition		
Section	**Variables**	**Constraints**
No Decompositions Goal or Softgoal		For a node i where Equation (5.2) has not been defined, and Given $\lvert M_{(i)}\rvert = 0$ and $i \in G \cup SG$ and $default_{(i)} = NaN$ define: $mVal_{(i)} = 0$ (5.3)
No Decompositions Non-Leaf Node		Given $\lvert M_{(i)}\rvert = 0$ and $\lvert C_{(i)} \cup D_{(i)}\rvert > 0$ define: $mVal_{(i)} = 0$ (5.4)
AND	$(0 \leq m_{(i.j)} \leq 100)$ $(0 \leq m'_{(i,j)} \leq 100)$	For every node j in $M_{(i)}$ where the relation is AND define: $mVal_{(i)} + m_{(i,j)} = dVal_{(j)}$ (5.5a) $m_{i,j} \cdot m'_{(i,j)} = 0$ (5.5b) Additionally, define the following constraint $\sum_{j \in M_i} \left(m'_{(i,j)}\right) \geq 1$ (5.6)
OR	$(0 \leq m_{(i.j)} \leq 100)$ $(0 \leq m'_{(i,j)} \leq 100)$	For every node j in $M_{(i)}$ where the relation is OR define: $mVal_{(i)} - m_{(i,j)} = dVal_{(j)}$ (5.7a) $m_{i,j} \cdot m'_{(i,j)} = 0$ (5.7b) Additionally, define the following constraint $\sum_{j \in M_i} \left(m'_{(i,j)}\right) \geq 1$ (5.8)

Table 5.4: Constraints and variable bounds required to model decomposition relationships in the alternative optimisation schema

Option	Equation	Generated Constraints
ALL	(5.5a)	$mVal_{Goal1} + m_{Goal1,Task1} = dVal_{Task1}$
ALL	(5.5a)	$mVal_{Goal1} + m_{Goal1,Task2} = dVal_{Task2}$
ALL	(5.6)	$m'_{Goal1,Task1} + m'_{Goal1,Task2} \geq 1$
ALL	(5.7a)	$mVal_{Goal2} + m_{Goal2,Task3} = dVal_{Task3}$
ALL	(5.7a)	$mVal_{Goal2} + m_{Goal2,Task4} = dVal_{Task4}$
ALL	(5.8)	$m'_{Goal2,Task3} + m'_{Goal2,Task4} \geq 1$
ALL	(5.7a)	$mVal_{Task1} + m_{Task1,Resource1} = dVal_{Resource1}$
ALL	(5.8)	$m'_{Task1,Resource1} \geq 1$
ALL	(5.4)	$mVal_{Task2} = 0$
ALL	(5.4)	$mVal_{Task4} = 0$

Table 5.5: The constraints necessary to model decomposition in the *demonstration graph*.

most. This difference can be seen in table 5.7. No capped value constraints must be generated, and only an upper tolerance is required.

Contribution and Correlation			
Section	**Variables**	**Constraints**	
None	$(0 \leq cVal_{(i)} \leq 100)$	if $\lvert C_{(i)} \rvert < 0$	
		$cVal_{(i)} = mVal_{(i)}$	(5.9)
Initial Score	$(-\infty \leq cVal2_{(i)} \leq \infty)$	$cVal2_{(i)} = \sum_{\substack{j \in C_{(i)} \\ (i,j) \in contrib}} \left(dVal_{(j)} \cdot \frac{contrib_{(i,j)}}{100} \right) + mVal_{(i)}$	(5.10)
Lower Limit	$(0 \leq cVal1_{(i)} \leq \infty)$ $(0 \leq n'_{(i)} \leq \infty)$ $(0 \leq p'_{(i)} \leq \infty)$	$cVal2_{(i)} + n'_(i) - p'_{(i)} = 0$	(5.11a)
		$cVal1_{(i)} = cVal2_{(i)} + n'_{(i)}$	(5.11b)
		$n'_{(i)}$ and $p'_{(i)} \geq 0$	(5.11c)
		$n'_{(i)} \cdot p'_{(i)} = 0$	(5.11d)
No Tolerance	$(0 \leq tol_{(i)} \leq 100)$	$tol_{(i)} = 100$	(5.12)
Tolerance	$(0 \leq nT_{(i)} \leq 100)$ $(0 \leq pT_{(i)} \leq 100)$ $(0 \leq inTol_{(i)} \leq 100)$	$mVal_{(i)} + nT_{(i)} - pT_{(i)} = inTol_{(i)}$	(5.13a)
		$tol_{(i)} = mVal_{(i)} + nT_{(i)}$	(5.13b)
		$nT_{(i)} \cdot pT_{(i)} = 0$	(5.13c)
Upper Limit	$(0 \leq n_{(i)} \leq 100)$ $(0 \leq p_{(i)} \leq 100)$	$cVal1_{(i)} + n_(i) - p_{(i)} = tol_{(i)}$	(5.14a)
		$cVal_{(i)} = cVal1_{(i)} - p_{(i)}$	(5.14b)
		$n_{(i)}$ and $p_{(i)} \geq 0$	(5.14c)
		$n_{(i)} \cdot p_{(i)} = 0$	(5.14d)

Table 5.6: Constraints and variable bounds required to model contribution and correlation relationships in the alternative optimisation schema

Option	Equation	Generated Constraints
ALL	(5.9)	$cVal_{Goal1} = mVal_{Goal1}$
ALL	(5.9)	$cVal_{Goal2} = mVal_{Goal2}$
ALL	(5.9)	$cVal_{Task1} = mVal_{Task1}$
ALL	(5.9)	$cVal_{Task2} = mVal_{Task2}$
ALL	(5.9)	$cVal_{Task3} = mVal_{Task3}$
ALL	(5.9)	$cVal_{Task5} = mVal_{Task5}$
ALL	(5.9)	$cVal_{Task6} = mVal_{Task6}$
ALL	(5.9)	$cVal_{Resource1} = mVal_{Resource1}$
ALL	(5.9)	$cVal_{Resource2} = mVal_{Resource2}$
ALL	(5.9)	$cVal_{Resource3} = mVal_{Resource3}$
ALL	(5.10)	$cVal2_{Task4} = (0.36 \times dVal_{Resource3}) + (0.42 \times dVal_{Task5}) + (-0.24 \times dVal_{Task6}) + mVal_{Task4}$
ALL	(5.11a)	$cVal2_{Task4} + n'_{Task4} - p'_{Task4} = 0$
ALL	(5.11b)	$cVal1_{Task4} = cVal2_{Task4} + n'_{Task4}$
¬Tol	(5.12)	$tol_{Task4} = 100$
Tol	(5.13a)	$mVal_{Task4} + nT_{Task4} - pT_{Task4} = inputTol_{Task4}$
Tol	(5.13b)	$tol_{Task4} = mVal_{Task4} + nT_{Task4}$
ALL	(5.11a)	$cVal1_{Task4} + n_{Task4} - p_{Task4} = tol_{Task4}$
ALL	(5.11b)	$cVal_{Task4} = cVal1_{Task4} - p_{Task4}$

Table 5.7: The constraints generated for the *demonstration graph* according to table 5.6.

5.1.4 Dependency Relationships

Table 5.8 shows the equations required to model dependency relationships.

Demonstration Graph Table 5.9 shows the constraints added to the optimisation model in order to enforce dependency relationships present in the *demonstration graph*.

5.1.5 Optimisation Model Comparison

The aim of the alternative optimisation schema was to provide a smaller optimisation model, resulting in faster computation times. Table 5.10 presents the complete optimisation model necessary for the *demonstration graph*, table 5.11 presents the equivalent optimisation model generated according to the original optimisation schema presented in chapter 4. The original optimisation schema required 62 constraints and ≈ 90 variables, in contrast the alternative schema requires only 37 constraints and ≈ 55 variables. The majority of optimisation solvers are matrix based solvers, so treating these models as matrices, the alternative model is less than half the size of the original model. This comparison outlines the differences in the two schemas, section 5.2.3 shall further compare the evaluation time of of the two schemas, based on actual data and not just model size.

Section	Variables	Dependency / Constraints	
No Relationships		$dVal_{(i)} = cVal_{(i)}$	(5.15)
Initial Calculation	$(0 \leq d_{(i,j)} \leq 100)$ $(0 \leq d'_{(i,j)} \leq 100)$	For every node j in $D_{(i)}$ define:	
		$dVal_{(i)} + d_{(i,j)} = dVal_{(j)}$	(5.16a)
		$d_{(i,j)}$ and $d'_{(i,j)} \geq 0$	(5.16b)
		$d_{(i,j)} \cdot d'_{(i,j)} = 0$	(5.16c)
		If $\lvert M_{(i)} \cup C_{(i)} \rvert = 0$ define:	
		$\sum_{j \in M_i} \left(d'_{(j)} \right) = 1$	(5.17)
Previous Relationships	$(0 \leq c_{(i)} \leq 100)$ $(0 \leq c'_{(i)} \leq 100)$	If $\lvert M_{(i)} \cup C_{(i)} \rvert > 0$ define:	
		$dVal_{(i)} + c_{(i)} = cVal_{(i)}$	(5.18a)
		$c_{(i)}$ and $c'_{(i)} \geq 0$	(5.18b)
		$c_i \cdot c'_{(i)} = 0$	(5.18c)
		If $\lvert M_{(i)} \cup C_{(i)} \rvert > 0$ replace equation (5.17) with:	
		$\sum_{j \in M_i} \left(d'_{(j)} \right) + c'_{(i)} = 1$	(5.19)

Table 5.8: Constraints and variable bounds required to model dependency relationships in the alternative optimisation schema

Option	Equation	Generated Constraints
ALL	(5.16a)	$dVal_{Task2} + d_{(Task2,Resource1)} = dVal_{Resource1}$
ALL	(5.16a)	$d'_{(Task2,Resource1)} = 1$
ALL	(5.15)	$dVal_{Goal1} = cVal_{Goal1}$
ALL	(5.15)	$dVal_{Goal2} = cVal_{Goal2}$
ALL	(5.15)	$dVal_{Task1} = cVal_{Task1}$
ALL	(5.15)	$dVal_{Task3} = cVal_{Task3}$
ALL	(5.15)	$dVal_{Task4} = cVal_{Task4}$
ALL	(5.15)	$dVal_{Task5} = cVal_{Task5}$
ALL	(5.15)	$dVal_{Task6} = cVal_{Task6}$
ALL	(5.15)	$dVal_{Resource1} = cVal_{Resource1}$
ALL	(5.15)	$dVal_{Resource2} = cVal_{Resource2}$
ALL	(5.15)	$dVal_{Resource3} = cVal_{Resource3}$

Table 5.9: The constraints generated to implement dependency relationships for the *demonstration graph*.

Table 5.11: The complete optimisation model generated for the *demonstration graph*, figure 4.2, according to the original optimisation schema, using only capped values.

Equation	Generated Constraints
(4.1)	$\max\left((0.5 \times dVal_{Goal1}) + (0.75 \times dVal_{Goal2})\right)$
(4.3a)	$mVal_{Goal1} + m_{Goal1,Task1} = dVal_{Task1}$
(4.3a)	$mVal_{Goal1} + m_{Goal1,Task2} = dVal_{Task2}$
(4.3d)	$m'_{Goal1,Task1} + m'_{Goal1,Task2} \geq 1$
(4.4a)	$mVal_{Goal2} + m_{Goal2,Task3} = dVal_{Task3}$
(4.4a)	$mVal_{Goal2} + m_{Goal2,Task4} = dVal_{Task4}$
(4.4d)	$m'_{Goal2,Task3} + m'_{Goal2,Task4} \geq 1$
(4.4a)	$mVal_{Task1} + m_{Task1,Resource1} = dVal_{Resource1}$
(4.4d)	$m'_{Task1,Resource1} \geq 1$
(4.6)	$mVal_{Task2} = 0$
(4.6)	$mVal_{Task4} = 0$
(4.7)	$cVal_{Goal1} = mVal_{Goal1}$
(4.7)	$cVal_{Goal2} = mVal_{Goal2}$
(4.7)	$cVal_{Task1} = mVal_{Task1}$

(4.7) $cVal_{Task2} = mVal_{Task2}$

(4.7) $cVal_{Task3} = mVal_{Task3}$

(4.7) $cVal_{Task5} = mVal_{Task5}$

(4.7) $cVal_{Task6} = mVal_{Task6}$

(4.7) $cVal_{Resource1} = mVal_{Resource1}$

(4.7) $cVal_{Resource2} = mVal_{Resource2}$

(4.7) $cVal_{Resource3} = mVal_{Resource3}$

(4.9a) $dVal_{Goal1} + x'_{Goal1} - x_{Goal1} = 0$

(4.9a) $dVal_{Goal2} + x'_{Goal2} - x_{Goal2} = 0$

(4.9a) $dVal_{Task1} + x'_{Task1} - x_{Task1} = 0$

(4.9a) $dVal_{Task2} + x'_{Task2} - x_{Task2} = 0$

(4.9a) $dVal_{Task3} + x'_{Task3} - x_{Task3} = 0$

(4.9a) $dVal_{Task4} + x'_{Task4} - x_{Task4} = 0$

(4.9a) $dVal_{Task5} + x'_{Task5} - x_{Task5} = 0$

(4.9a) $dVal_{Task6} + x'_{Task6} - x_{Task6} = 0$

(4.9a) $dVal_{Resource1} + x'_{Resource1} - x_{Resource1} = 0$

(4.9a) $dVal_{Resource2} + x'_{Resource2} - x_{Resource2} = 0$

(4.9a) $dVal_{Resource3} + x'_{Resource3} - x_{Resource3} = 0$

(4.9b) $capVal_{Goal1} = dVal_{Goal1} + x'_{Goal1}$

(4.9b) $capVal_{Goal2} = dVal_{Goal2} + x'_{Goal2}$

(4.9b) $capVal_{Task1} = dVal_{Task1} + x'_{Task1}$

(4.9b) $capVal_{Task2} = dVal_{Task2} + x'_{Task2}$

(4.9b) $capVal_{Task3} = dVal_{Task3} + x'_{Task3}$

(4.9b) $capVal_{Task4} = dVal_{Task4} + x'_{Task4}$

(4.9b) $capVal_{Task5} = dVal_{Task5} + x'_{Task5}$

(4.9b) $capVal_{Task6} = dVal_{Task6} + x'_{Task6}$

(4.9b) $capVal_{Resource1} = dVal_{Resource1} + x'_{Resource1}$

(4.9b) $capVal_{Resource2} = dVal_{Resource2} + x'_{Resource2}$

(4.9b) $capVal_{Resource3} = dVal_{Resource3} + x'_{Resource3}$

(4.8) $cVal2_{Task4} = (0.36 \times capVal_{Resource3}) + (0.42 \times capVal_{Task5}) + (-0.24 \times capVal_{Task6}) + mVal_{Task4}$

(4.11) $nTol_{Task4} = -100$

(4.13a) $cVal2_{Task4} + n'_{Task4} - p'_{Task4} = nTol_{Task4}$

(4.13b) $cVal1_{Task4} = cVal2_{Task4} + n'_{Task4}$

(4.14)	$tol_{Task4} = 100$
(4.15a)	$mVal_{Task4} + nT_{Task4} - pT_{Task4} = inputTol_{Task4}$
(4.15b)	$tol_{Task4} = mVal_{Task4} + nT_{Task4}$
(4.13a)	$cVal1_{Task4} + n_{Task4} - p_{Task4} = tol_{Task4}$
(4.13b)	$cVal_{Task4} = cVal1_{Task4} - p_{Task4}$
(4.17a)	$dVal_{Task2} + d_{(Task2,Resource1)} = dVal_{Resource1}$
(4.17a)	$d'_{(Task2,Resource1)} = 1$
(4.21)	$dVal_{Goal1} = cVal_{Goal1}$
(4.21)	$dVal_{Goal2} = cVal_{Goal2}$
(4.21)	$dVal_{Task1} = cVal_{Task1}$
(4.21)	$dVal_{Task3} = cVal_{Task3}$
(4.21)	$dVal_{Task4} = cVal_{Task4}$
(4.21)	$dVal_{Task5} = cVal_{Task5}$
(4.21)	$dVal_{Task6} = cVal_{Task6}$
(4.21)	$dVal_{Resource1} = cVal_{Resource1}$
(4.21)	$dVal_{Resource2} = cVal_{Resource2}$
(4.21)	$dVal_{Resource3} = cVal_{Resource3}$

Equation	Generated Constraints
(5.1)	$\max((0.5 \times dVal_{Goal1}) + (0.75 \times dVal_{Goal2}))$
(5.5a)	$mVal_{Goal1} + m_{Goal1,Task1} = dVal_{Task1}$
(5.5a)	$mVal_{Goal1} + m_{Goal1,Task2} = dVal_{Task2}$
(5.6)	$m'_{Goal1,Task1} + m'_{Goal1,Task2} \geq 1$
(5.7a)	$mVal_{Goal2} + m_{Goal2,Task3} = dVal_{Task3}$
(5.7a)	$mVal_{Goal2} + m_{Goal2,Task4} = dVal_{Task4}$
(5.8)	$m'_{Goal2,Task3} + m'_{Goal2,Task4} \geq 1$
(5.7a)	$mVal_{Task1} + m_{Task1,Resource1} = dVal_{Resource1}$
(5.8)	$m'_{Task1,Resource1} \geq 1$
(5.4)	$mVal_{Task2} = 0$
(5.4)	$mVal_{Task4} = 0$
(5.9)	$cVal_{Goal1} = mVal_{Goal1}$
(5.9)	$cVal_{Goal2} = mVal_{Goal2}$
(5.9)	$cVal_{Task1} = mVal_{Task1}$
(5.9)	$cVal_{Task2} = mVal_{Task2}$
(5.9)	$cVal_{Task3} = mVal_{Task3}$
(5.9)	$cVal_{Task5} = mVal_{Task5}$
(5.9)	$cVal_{Task6} = mVal_{Task6}$
(5.9)	$cVal_{Resource1} = mVal_{Resource1}$
(5.9)	$cVal_{Resource2} = mVal_{Resource2}$
(5.9)	$cVal_{Resource3} = mVal_{Resource3}$
(5.10)	$cVal2_{Task4} = (0.36 \times dVal_{Resource3}) + (0.42 \times dVal_{Task5}) + (-0.24 \times dVal_{Task6}) + mVal_{Task4}$
(5.11a)	$cVal2_{Task4} + n'_{Task4} - p'_{Task4} = 0$
(5.11b)	$cVal1_{Task4} = cVal2_{Task4} + n'_{Task4}$
(5.12)	$tol_{Task4} = 100$
(5.11a)	$cVal1_{Task4} + n_{Task4} - p_{Task4} = tol_{Task4}$
(5.11b)	$cVal_{Task4} = cVal1_{Task4} - p_{Task4}$
(5.16a)	$dVal_{Task2} + d_{(Task2,Resource1)} = dVal_{Resource1}$
(5.16a)	$d'_{(Task2,Resource1)} = 1$
(5.9)	$dVal_{Goal1} = cVal_{Goal1}$
(5.9)	$dVal_{Goal2} = cVal_{Goal2}$
(5.9)	$dVal_{Task1} = cVal_{Task1}$
(5.9)	$dVal_{Task3} = cVal_{Task3}$
(5.9)	$dVal_{Task4} = cVal_{Task4}$
(5.9)	$dVal_{Task5} = cVal_{Task5}$
(5.9)	$dVal_{Task6} = cVal_{Task6}$
(5.9)	$dVal_{Resource1} = cVal_{Resource1}$
(5.9)	$dVal_{Resource2} = cVal_{Resource2}$
(5.9)	$dVal_{Resource3} = cVal_{Resource3}$

Table 5.10: The optimisation model generated for figure 5.3, according to the alternative optimisation schema, using neither initialisation values or tolerance limits.

5.2 Test Case Application

This section applies the alternative optimisation schema to the *Telecommunication Exemplar* and *NetME Plugin* test cases, as covered in section 4.4. The discussion analyses any result differentials and compares the computation times. Section 4.4.2 explained how the *NetME Interface* did not model negative contributions, as such it was not possible for any nodes to take a negative value, given this situation, the results of the alternative optimisation schema are redundant and not discussed. The NetME Interface has, however, been included in section 5.2.3 in order to compare the computation times of the two optimisation schemas.

5.2.1 Telecommunication Exemplar

Default: Once again, the smaller of the test cases, the *Telecommunication Exemplar*, is the first to have the optimisation schema applied. Figure 5.4 shows the results of the alternative optimisation schema without applying either initialisation values or tolerance evaluation, the final value of the objective function for this evaluation was 203.125.

Initialisation: Section 4.4.1.1 discussed how *Maximum Hardware Utilisation* had a major impact on the results of the graph and how initialisation could be used to incorporate previously disjoint data. Figure 5.5 shows the updated optimisation results when initialisation is applied to the highlighted node, in this evaluation the final value of the objective function was 254.6875.

The effect of initialisation on the *Telecommunication Exemplar* is, as expected, the same as the original optimisation schema. *High Performance* has a marked improvement, whereas *Low Cost* only has a slight improvement. Again, due to the structure of the graph, use of tolerance evaluation would not alter the provided results. The alternative optimisation schema was designed to decrease the complexity of generated models, that is, its focus is to offer a *time* improvement. As such, there

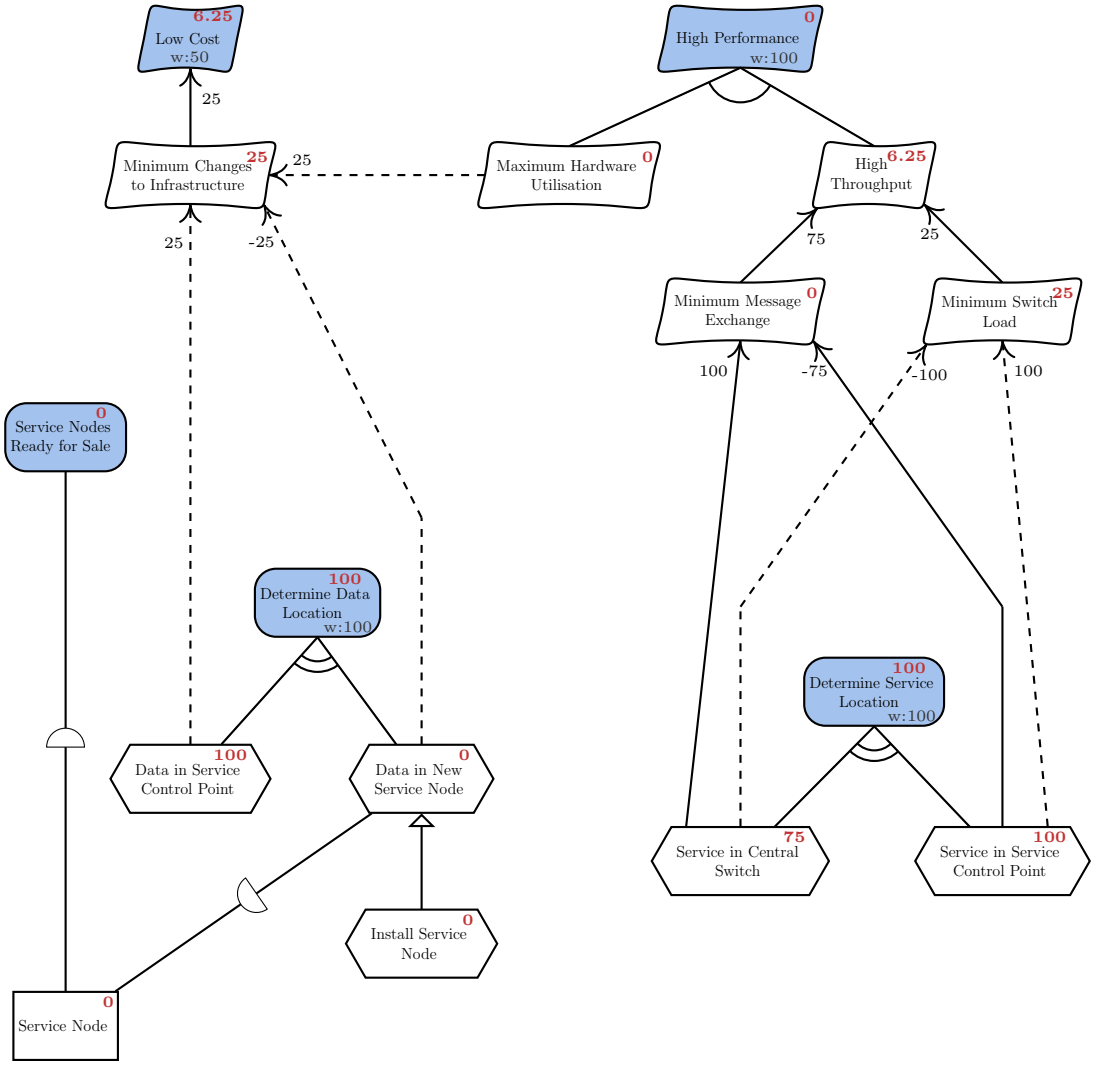


Figure 5.4: Results of the alternative optimisation schema as applied to the Telecommunication Exemplar.

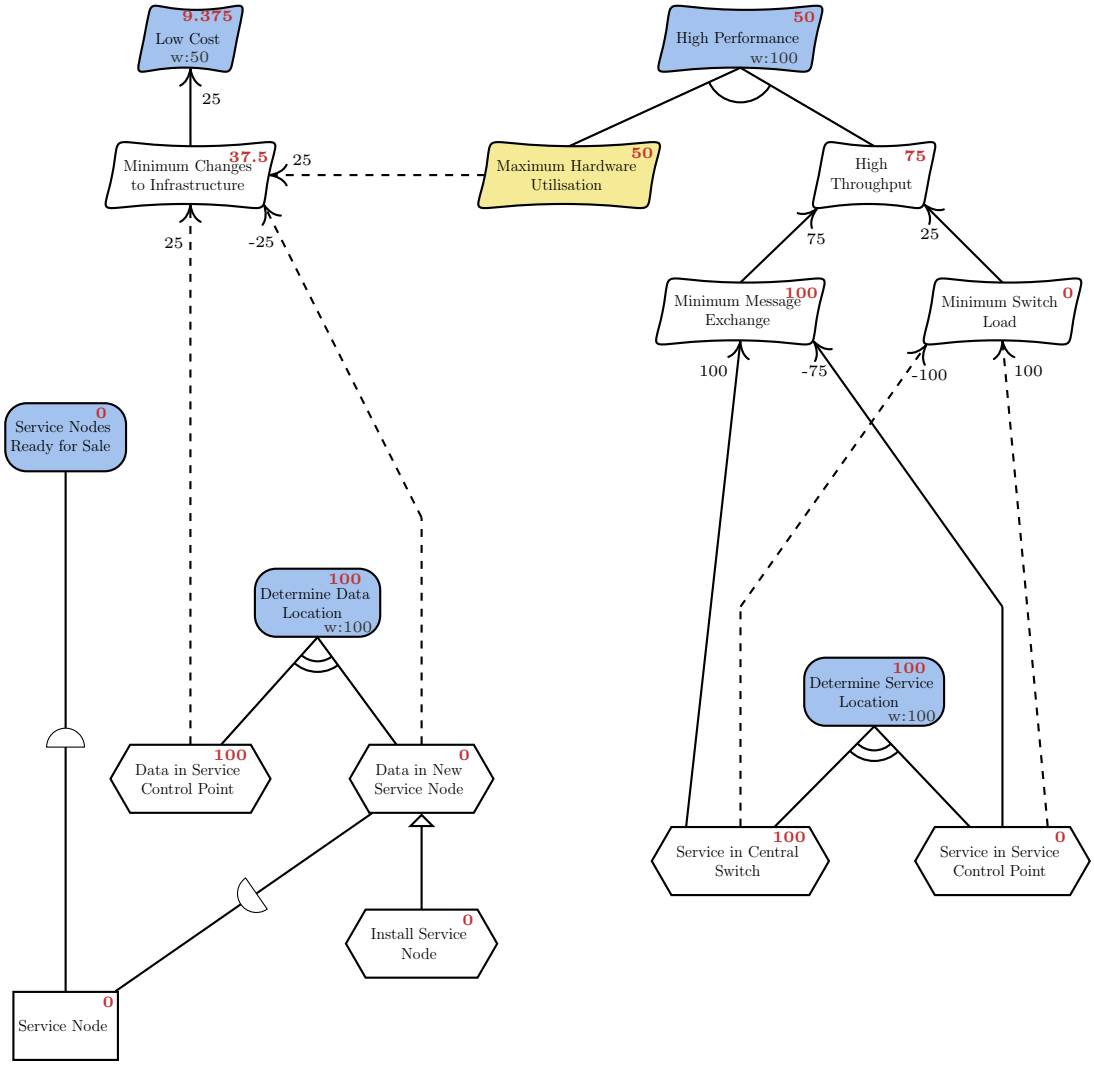


Figure 5.5: Results of the alternative optimisation schema utilising initialisation values as applied to the Telecommunication Exemplar.

is very little change in the results of the optimisation, the value of the objective function in both the provided graphs is identical to their original counterparts. Due to the fact that nodes can no longer take negative values one minor change appeared in the graph, *Minimum Switch Load* now has a value of 0, however, this change does not affect either the value of the leaf nodes or the root nodes. Furthermore, the change is expected due to the modified paradigm of the alternative optimisation schema.

5.2.2 NetME Plugin

Section 4.4.3 applied the original optimisation schema with various evaluation options to the *NetME Plugin*, in this process the computation time for the *capped values* option became unreasonable.

Default: Figure 5.6 shows the results of the alternative optimisation schema without using initialisation or tolerance, this would be comparable to the original optimisation schema using *only* capped values. The final value of the objective function in this evaluation was 349.09.

Initialisation: Figure 5.7 applies initialisation values to the graph, these values differ as nodes that previously held a value of -100 now take a zero value, once again these nodes are highlighted for clarity. The final value of the objective function this time is 299.09. As expected, there are several key differences in the graph, most notable are the contributors to *Space Performance* and *Time Performance*. *Display During Execution* no longer has an affect on *Time Performance*, this means that *Check On Insert* must take a non-zero value in order to ensure both *Time Performance* and its parent *Performance* take a non-zero value. Consequently, *On-Insert Warning* must also take a non-zero value resulting in *Specify On Insert* to take a lower value.

Initialisation & Tolerance: Figure 5.8 shows the results of the alternative optimisation schema with both initialisation values and tolerance evaluation. This graph is the direct counterpart to figure 4.17 and as expected the value of the objective function is the same in both cases — 289.09 — the only noticeable change is *Specify On Insert* now has a zero value. Due to tolerance evaluation, the value of this node is no longer required to ensure *Usability* reaches its optimal score of 100.

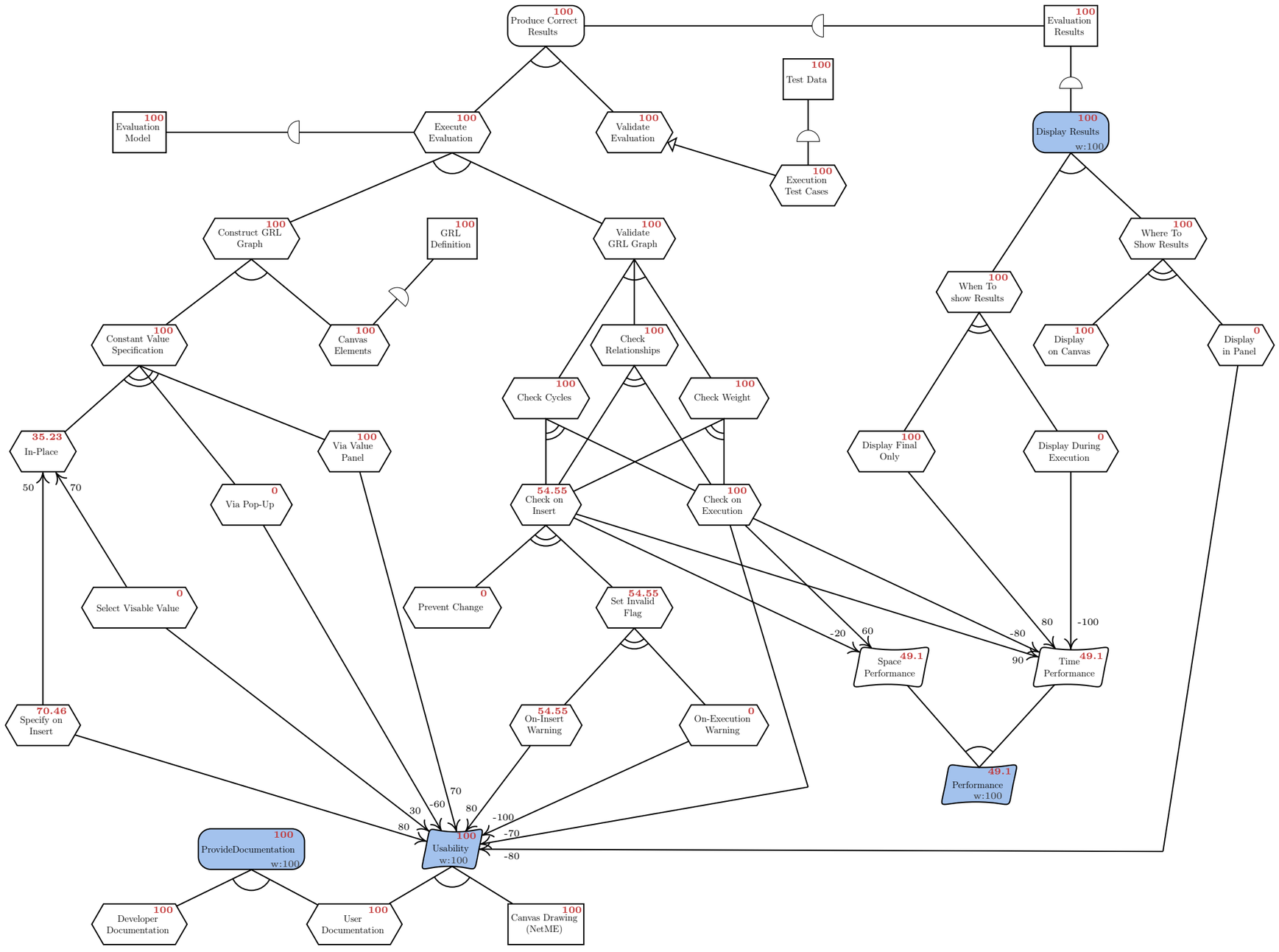


Figure 5.6: Results of the alternative optimisation schema as applied to the NetME Plugin.

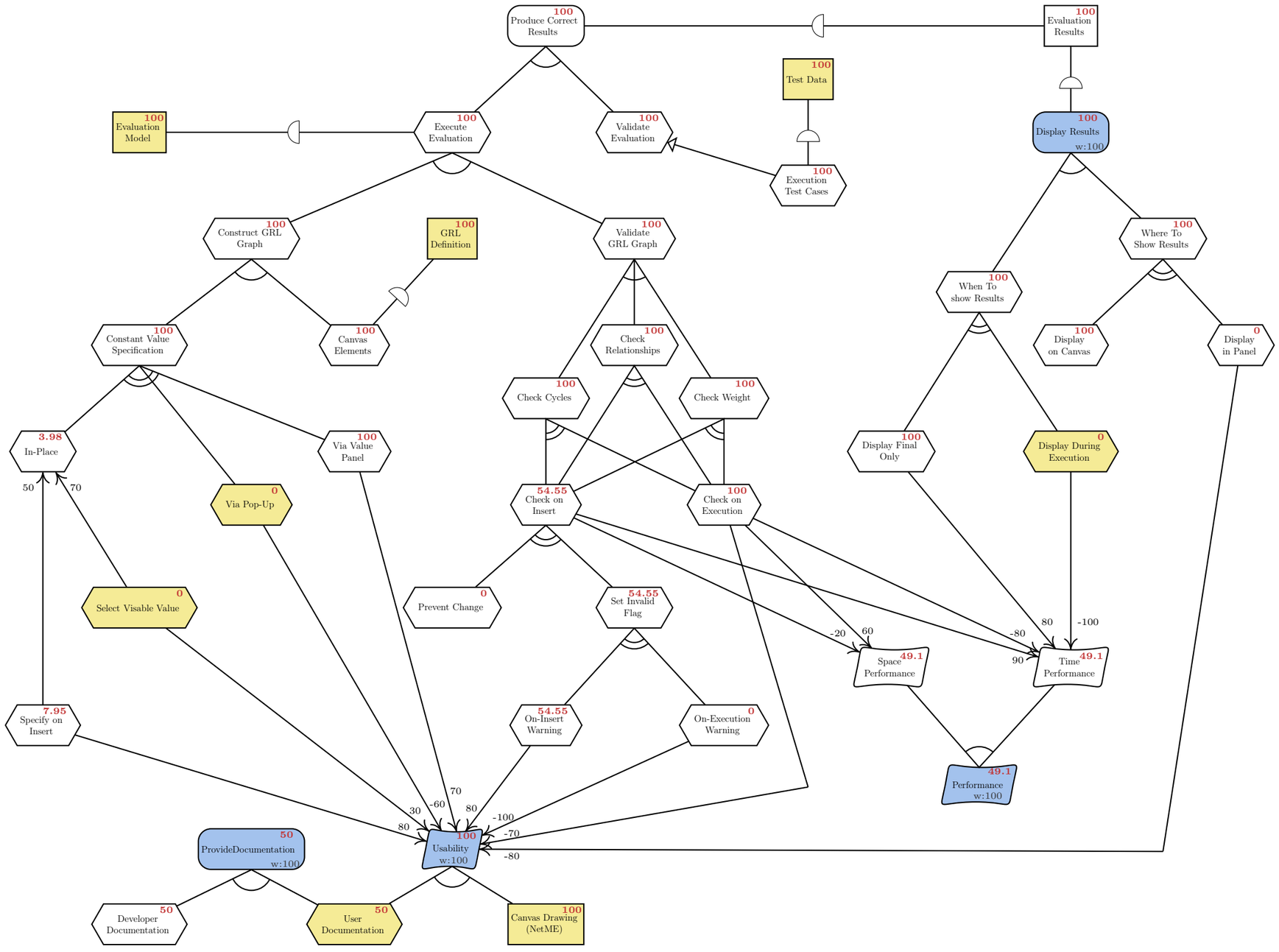


Figure 5.7: Results of the alternative optimisation schema with use of initialisation values, as applied to the NetME Plugin.

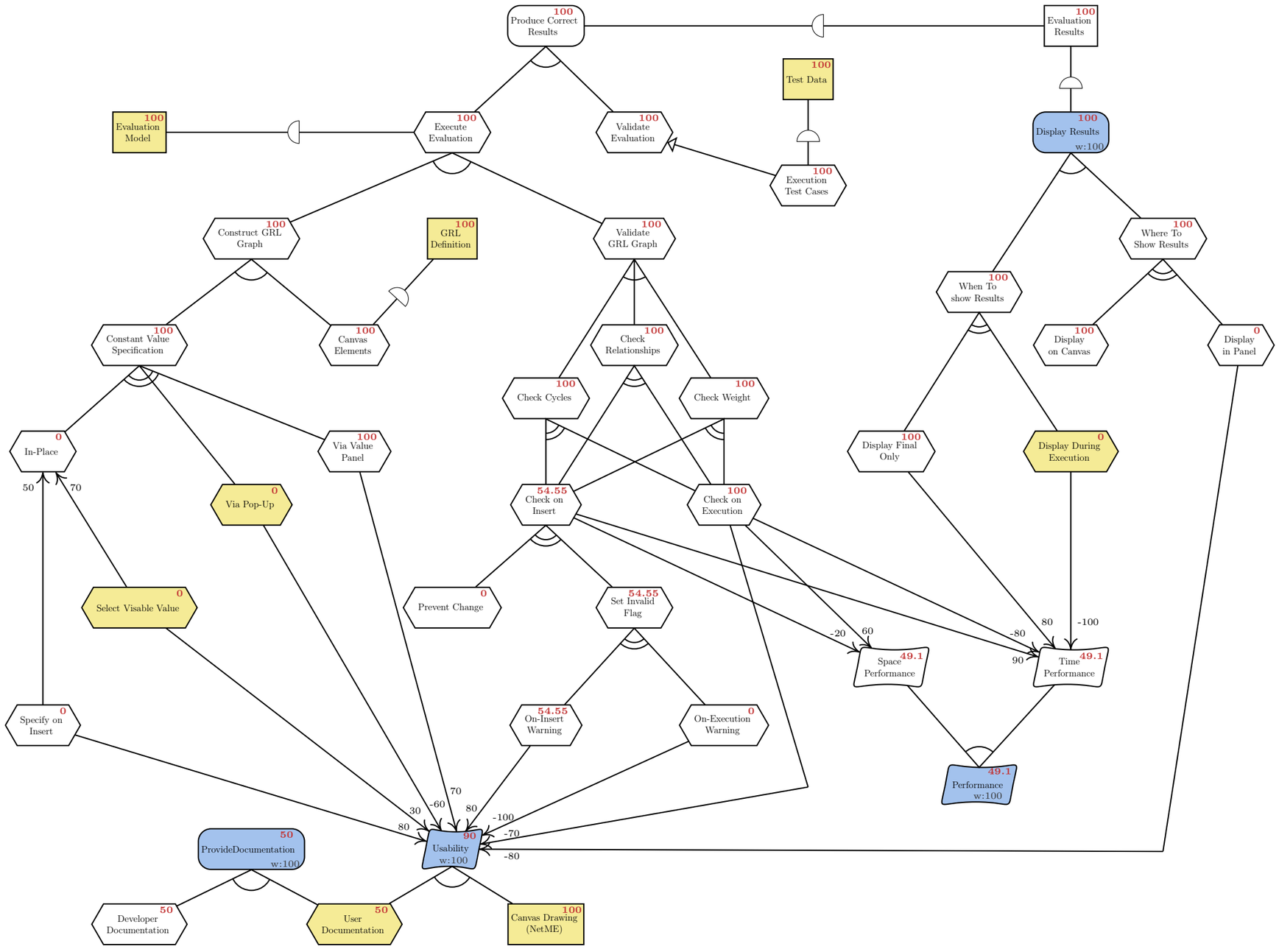


Figure 5.8: Results of the alternative optimisation schema with use of initialisation values and tolerance evaluation, as applied to the NetME Plugin.

Evaluation Option	Paper Graph		NetME Interface		NetME Plugin	
Default	$0.49s$	$(2.62s)$	$0.032s$	$(0.032s)$	$0.063s$	$(> 24h)$
Initialisation	$< 0.01s$	$(< 0.01s)$	$0.032s$	$(0.031s)$	$0.063s$	$(> 24h)$
Tolerance	$0.151s$	$(2.933s)$	$0.047s$	$(0.063s)$	$2.593s$	$(> 24h)$
Initialisation & Tolerance	$< 0.01s$	$(0.015s)$	$0.031s$	$(0.031s)$	$2.262s$	$(> 24h)$

Table 5.12: Time comparisons of the evaluation options for the alternate optimisation schema across the three test cases.

5.2.3 Time Analysis

Section 4.4.4 demonstrated that the original optimisation schema had eight different possible evaluations, as the alternative optimisation schema removes one of these options, it only has four possibilities. Table 5.12 shows the computation time required for each of these options for each test case; the NetME Interface has been included here to show evaluation times, even though a detailed analysis of its results were deemed redundant. The evaluation times of the equivalent strategy in the original optimisation schema have been provided — in brackets — accompanying each evaluation time.

This comparison clearly shows a marked improvement in computation time in all situations where the *capped values* options had an effect. In the extreme cases the computation time goes from several hours to only a few seconds, but even when the original computation was measured in seconds, the alternative optimisation schema still out performed the original by a considerable *percentage*. This drastic improvement has a direct correlation with the size difference of the generated models, as discussed in section 5.1.5.

5.3 Lessons Learned

This chapter largely implemented the lessons learned from previous chapters, however, this time considerable attention was paid to the relationship between constraints and time complexity. Specifically, that the increase in size[1] complexity of $\mathcal{O}(n)$ — specifically $\mathcal{O}(2n+1)$ constraints and $\mathcal{O}(3n)$ limits — results in an exponential — $\mathcal{O}(2^n)$ — effect on time complexity. From this understanding it became apparent that the non-trivial constraints necessary to model the *capped value* version of the optimisation schema were causing a serious problem for larger values of n. This issue was overcome by removing the necessity for additional constraints, specifically, every score calculation inherently fell within the capped value range.

5.4 Summary

This chapter presented an alternative optimisation schema for the goal-oriented requirements language based on the notion of *only* modelling the effect a node's existence has on achieving the root goals. This alteration resulted in the merger of several schema constraints, as specified in section 5.1. The alternative optimisation schema was applied to several test cases and the results analysed and compared to the previous chapter; the computation time analysis and comparison showed a marked improvement when compared to the equivalent evaluation choices of the original optimisation schema.

The next chapter presents a summary of the work completed in this thesis and discusses how the four research objectives were achieved. Additionally, the chapter will present several ideas for future work based on the discoveries of previous chapters.

[1] For this purpose size is determined by the number of nodes n comprising the graph.

Chapter 6

Conclusion and Future Work

This thesis explored the possibility of applying mathematical optimisation to goal-oriented requirements engineering graph models in order to investigate the possibility of improving current evaluation techniques. Section 2.6 provided readers with a brief introduction to mathematical optimisation and explained how to apply the presented optimisation schemas.

Chapter 3 developed the first optimisation schema and applied it to the Non-Functional Requirements framework. This chapter included a novel approach to conducting a sensitivity analysis, allowing the use of flexible quantitative values, an ongoing issues in any quantitative approach. Sections 2.6.2.1 and 3.5 showed how the interpretation of sensitivity analysis results can not only overcome the issues associated with quantitative values, but also be utilised by a development team in conflict resolution. Chapter 4 investigated the application of optimisation to the goal-orientated requirements language, a far more flexible modelling tool than the NFR Framework. The schema developed in this chapter included several evaluation strategies that maintained the flexibility of the framework and developer control. Chapter 5 investigated issues with computation time and developed an optimisation schema that addressed and overcame them. All of the developed optimisation schemas were also applied to several test cases in order to determine their effectiveness and assess their computation time.

6.1 Research Objectives

Chapter 1, section 1.1, outlined several research objectives that would be explored in this thesis. This section will discuss these objectives in terms of the work presented in previous chapters.

Objective 1: Decision Aid

> *Can evaluation methods be developed that accurately* recommend *design decisions rather than just* assess *their effectiveness?*

From the outset, the goal was to develop evaluation techniques that did not just propagate predetermined decisions. By implementing mathematical optimisation the proposed methods actually *decided* which techniques/tasks should be implemented. Additionally, the ability to incorporate developers decisions in value *propagation*, or through customised optimisation strategies, allowed developers to maintain control over the results.

Objective 2: Overcome Scalability Issues and Computation Time

> *Can the proposed methods be applied to graphs of considerable size without suffering degradation, either in their accuracy or computation time?*

Chapters 1 and 2 explained the scalability issues present in current evaluation techniques, as such, a major objective for the research was to create a method that would not suffer from scalability issues. To assess this objective, each of the optimisation schemas was applied to graphs of considerable size. Evaluation of these graphs produced consistent accurate results, additionally, degradation of computation time only occurred on graphs of inordinate proportions.

Objective 3: Produce an Optimal Result

Do the proposed methods confidently and consistently produce results that cannot be improved?

To ensure that the results truly provided the best possible solution, the developed methods relied on mathematical optimisation. Inherently, optimisation should provide the result that produces the highest possible value for a given graph structure. To ensure that this property was not broken, the schema verification included manually generating alternatives in an attempt to achieve a higher result than that provided by the optimisation; appendix A demonstrated part of this process. This process revealed that the optimisation always produced the highest scoring results — though at times this score could be achieved through multiple node combinations.

Objective 4: Automation

Can the proposed methods be calculated with minimal developer input?

Evaluation of an optimisation model requires the use of an optimisation solver. However, in order for optimisation to provide a feasible evaluation method it must be possible to automatically generate a model from any given graph. Chapters 3–5 implemented procedural algorithms that generated individual optimisation models, used an optimisation solver to generate results, and then linked the results back to their appropriate nodes. While these algorithms used text files as the data input, once the graphs were built the data input was no longer required, proving that regardless of the data origin, an optimisation model can be successfully generated and evaluated.

Bonus Objective: Conflict Resolution The development of a process that could aid in conflict resolution was not a set goal of this work. However, in investigating mathematical optimisation and consequently sensitivity analyses, it was determined that if a sensitivity analysis could be adapted to a goal graph, the results it produced would fill this need. Chapter 3, sections 3.4 and 3.5 developed and

applied a sensitivity analysis in this manner. Section 6.2.3 proposes further work related to this discovery.

6.2 Future Work

In developing the methods presented in this thesis, several different avenues of continued research became apparent.

6.2.1 Multi-Objective Optimisation

When applying an optimisation schema, the results are guaranteed to produce the highest possible value of the objective function; in the situations previously presented, that means the highest achievement of the system goals. However, what if there are two sets of results that would produce the equal levels of satisfaction. Application of a sensitivity analysis allows a snapshot of these alternatives to be viewed simultaneously. An alternative approach would be to apply a second iteration of optimisation to narrow down the choice selection. Various possibilities include minimising cost, minimising effort, or minimising the number of leaf-nodes requiring non-zero values.

6.2.2 Language Extension

An alternative approach to dealing with the various ideal solutions would be to develop a language or modelling extension capable of easily representing the various results. The solution would need to be quickly understandable with little additional reference and without excessive training. A solution to this problem would need to be innovative and require a significant user based testing.

6.2.3 Sensitivity Analysis Application for Non-Binary Leaf-Nodes

Chapter 3 developed an approach to performing a sensitivity analysis that allowed it to be applied in situations where changes to the objective function value are inconsequential. However, the approach required that the leaf-nodes be limited to binary values in order to have concrete evidence of a change. This means the approach was unable to be applied to either of the GRL schemas due to the fundamental approach of modelling the degree to which *every* node was satisfied, an approach necessary in order to accommodate the flexibility of the language. Future research could investigate further adaptations to the sensitivity analysis in order to apply it to situations with both an inconsequential objective function *and* non-binary leaf-nodes. Alternatively, an optimisation schema could be developed that includes some form of measurable change for tasks and resources.

6.2.4 Additional Applications

The NFR Framework provides a rigid modelling approach while GRL provides a flexible means to expressing any combination of approaches. However, the methods developed in this thesis could be expanded and adapted in order to create optimisation schemas for any number of GORE approaches or even other weighted node graphs.

Bibliography

[1] M. D. McIlroy, ""Mass produced" software components," in *Software Engineering*, P. Naur and B. Randell, Eds. Brussels: Scientific Affairs Division, NATO, , Garmisch, Germany, October 1969, pp. 138–155.

[2] T. E. Bell and T. A. Thayer, "Software requirements: Are they really a problem?" in *Proceedings of the 2nd international conference on Software engineering*, ser. ICSE '76. IEEE Computer Society Press, Los Alamitos, CA, USA, October 1976, pp. 61–68.

[3] B. W. Boehm, "Software engineering economics," *Software Engineering, IEEE Transactions on*, vol. 10, no. 1, pp. 4 –21, 1984.

[4] F. P. Brooks, "No silver bullet: Essence and accidents of software engineering," *IEEE Computer*, vol. 20, pp. 10–19, 1987.

[5] The Standish Group, ""Extreme Chaos"," www.standishgroup.com/chaos.html, 2000.

[6] E. Bjarnason, M. Unterkalmsteiner, M. Borg, and E. Engström, "A multi-case study of agile requirements engineering and the use of test cases as requirements," *Information and Software Technology*, vol. 77, pp. 61–79, 2016.

[7] L. A. Maciaszek, *Requirements Analysis and System Design*. Addison Wesley, 2007.

[8] D. Ross and J. Schoman, K.E., "Structured analysis for requirements definition," *Software Engineering, IEEE Transactions on*, vol. 3, no. 1, pp. 6 – 15, jan. 1977.

[9] S. Ouhbi, A. Idri, J. L. Fernández-Alemán, and A. Toval, "Requirements engineering education: a systematic mapping study," *Requirements Engineering*, vol. 20, no. 2, pp. 119–138, 2015.

[10] P. Zave, "Classification of research efforts in requirements engineering," *ACM Comput. Surv.*, vol. 29, pp. 315–321, December 1997.

[11] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 35–46.

[12] D. Mairiza, D. Zowghi, and N. Nurmuliani, "An investigation into the notion of non-functional requirements," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10. Sierre, Switzerland: ACM, New York, NY, USA, March 2010, pp. 311–317.

[13] A. Babiker and A.-E.-K. Sahraoui, "On non functional requirements, their evolution and impact on safety," in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016, p. 31.

[14] L. M. Cysneiros and J. C. S. do Prado Leite, "Using uml to reflect non-functional requirements," in *Proceedings of the 2001 Conference of the Centre for Advanced Studies on Collaborative Research*, 2001.

[15] L. M. Cysneiros, J. C. S. do Prado Leite, and J. de Melo Sabat Neto, "A framework for integrating non-functional requirements into conceptual models," *Requirements Engineering*, vol. 6, pp. 97–115, 2001, 10.1007/s007660170008.

[16] L. M. Cysneiros and J. C. S. do Prado Leite, "Nonfunctional requirements: From elicitation to conceptual models," *IEEE Transactions on Software Engineering*, vol. 30, pp. 328–350, 2004.

[17] M. Glinz, "On non-functional requirements," in *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, 2007, pp. 21 –26.

[18] A. van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. Toronto, Canada: IEEE Computer Society Press, Los Alamitos, CA, USA, August 2001, pp. 249–262.

[19] A. van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," *Software Engineering, IEEE Transactions on*, vol. 26, no. 10, pp. 978 –1005, Oct. 2000.

[20] A. van Lamsweerde, "Goal-oriented requirements enginering: a roundtrip from research to practice [enginering read engineering]," in *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*. Kyoto, Japan: IEEE Computer Press, Los Alamitos, CA, USA, Sept 2004, pp. 4 – 7.

[21] J. Mylopoulos, L. Chung, and E. Yu, "From object-oriented to goal-oriented requirements analysis," *Communications of the ACM*, vol. 42, no. 1, pp. 31–37, 1999.

[22] J. Horkoff and E. Yu, "Interactive goal model analysis for early requirements engineering," *Requirements Engineering*, vol. 21, no. 1, pp. 29–61, 2016.

[23] E. Yu, "Modelling strategic relationships for process reengineering," Ph.D. dissertation, Department of Computer Science, University of Toronto, Toronto, Canada, 1995.

[24] A. Dardenne, S. Fickas, and A. van Lamsweerde, "Goal-directed concept acquisition in requirements elicitation," in *Proceedings of the 6th international*

workshop on Software specification and design, ser. IWSSD '91. Como, Italy: IEEE Computer Society Press, Los Alamitos, CA, USA, October 1991, pp. 14–21.

[25] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[26] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, pp. 203–236, 2004.

[27] I. Jureta, A. Borgida, N. Ernst, and J. Mylopoulos, "Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*. Sydnew, NSW: IEEE, September 2010, pp. 115–124.

[28] S. Liaskos, S. Hamidi, and R. Jalman, "Qualitative vs. quantitative contribution labels in goal models: Setting an experimental agenda," in *iStar*, vol. 978. Valencia, Spain: Ceur Workshop Proceedings, June 2013, pp. 37–42.

[29] J. Castro, J. Horkoff, N. Maiden, and E. Yu, Eds., *iStar 2013, 6th interational i* Workshop*, vol. 978. Valencie, Spain: Ceur Workshop Proceedings,, June 2013.

[30] O. Akhigbe, M. Alhaj, D. Amyot, O. Badreddin, E. Braun, N. Cartwright, G. Richards, and G. Mussbacher, *Conceptual Modelling*. Cham: Springer International Publishing, 2014, ch. Creating Quantitative Goal Models: Governmental Experience, pp. 466–473.

[31] N. Kobayashi, S. Morisaki, N. Atsumi, and S. Yamamoto, "Quantitative non functional requirements evaluation using softgoal weight," *Journal of Internet Services and Information Security (JISIS)*, vol. 6, pp. 37–46, 2016.

[32] S. Yamamoto, *An Approach for Evaluating Softgoals Using Weight*. Cham: Springer International Publishing, 2015, pp. 203–212.

[33] A. Affleck and A. Krishna, "Supporting quantitative reasoning of non-functional requirements: A process-oriented approach," in *Software and System Process (ICSSP), 2012 International Conference on*. Zurich, Switzerland: IEEE Computer Press, Los Alamitos, CA, USA, June 2012, pp. 88 –92.

[34] L. Lopez and Y. Yu, Eds., *iStar 2016, 9th interational i* Workshop*, vol. 978, Beijing, china, September 2016.

[35] E. Letier and A. van Lamsweerde, "Reasoning about partial goal satisfaction for requirements and design engineering," *SIGSOFT Softw. Eng. Notes*, vol. 29, pp. 53–62, October 2004.

[36] A. van Lamsweerde, "Reasoning about alternative requirements options," in *Conceptual Modeling: Foundations and Applications*, ser. Lecture Notes in Computer Science, A. Borgida, V. Chaudhri, P. Giorgini, and E. Yu, Eds. Springer Berlin / Heidelberg, 2009, vol. 5600, pp. 380–397.

[37] L. Chung and B. A. Nixon, "Dealing with non-functional requirements: three experimental studies of a process-oriented approach," in *Proceedings of the 17th international conference on Software engineering*, ser. ICSE '95. Seattle, WA, USA: AC, New York, NY, USA, 1995, pp. 25–37.

[38] D. Amyot and G. Mussbacher, *URN: Towards a New Standard for the Visual Description of Requirements*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 21–37.

[39] R. Z. ITU-T, "150 (02/03): User requirements notation (urn)–language requirements and framework," *Geneva, Switzerland*, vol. 200337, 2003.

[40] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with goal models," in *Conceptual Modeling ER 2002*, ser. Lecture Notes in Computer Science, S. Spaccapietra, S. March, and Y. Kambayashi, Eds. Springer Berlin / Heidelberg, 2003, vol. 2503, pp. 167–181.

[41] S. Liaskos, S. McIlraith, S. Sohrabi, and J. Mylopoulos, "Integrating preferences into goal models for requirements engineering," in *Requirements Engineering Conference, 2010 18th IEEE International*. IEEE Computer Society, September 2010, pp. 135–144.

[42] A. Cailliau and A. van Lamsweerde, "A probabilistic framework for goal-oriented risk analysis," in *Requirements Engineering Conference (RE), 2012 20th IEEE International*. Chicago, Illinois: IEEE Computer Society, Washington, DC, USA, September 2012, pp. 201–210.

[43] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu, "Evaluating goal models within the goal-oriented requirement language," *International Journal of Intelligent Systems*, vol. 25, no. 8, pp. 841–877, 2010.

[44] W. Heaven and E. Letier, "Simulating and optimising design decisions in goal models," in *Proceedings of 19th IEEE International Requirements Engineering Conference (RE 2011)*. Trento, Italy: IEEE Computer Press, Los Alamitos, CA, USA, August 2011, pp. 79–88.

[45] B. Wei, Z. Jin, and D. Zowghi, "An automatic reasoning mechanism for nfr goal models," in *Theoretical Aspects of Software Engineering (TASE), 2011 Fifth International Symposium on*, aug. 2011, pp. 52 –59.

[46] B. Wei, Z. Jin, D. Zowghi, and B. Yin, "Automated reasoning with goal tree models for software quality requirements," in *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*, july 2012, pp. 373 –378.

[47] R. Degiovanni, D. Alrajeh, N. Aguirre, and S. Uchitel, "Automated goal operationalisation based on interpolation and sat solving," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 129–139.

[48] A. Mansoor, D. Streitferdt, and F.-F. Füßl, "Alternatives selection using gore based on fuzzy numbers and topsis," *Journal of Software Engineering and Applications*, vol. 8, no. 07, p. 346, 2015.

[49] S. Liaskos, R. Jalman, and J. Aranda, "On eliciting contribution measures in goal models," in *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE'12)*. Chicago, IL: Springer-Verlag New York, Inc. Secaucus, NJ, USA, September 2012, pp. 221–230.

[50] L. A. Johnson and D. C. Montgomery, *Operations Research in Production Planning, Scheduling and Inventory Control*. John Wiley & Sons Inc, 1974.

[51] M. Sasieni, A. Yaspan, and L. Friedman, *Operations Research: Methods and Problems*. John Wiley & Sons Inc, 1959.

[52] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[53] R. W. Shepherd, *Theory of Cost and Production Functions*. Princeton University Press, 2015.

[54] R. J. Vanderbei *et al.*, *Linear programming*. Springer, 2015.

[55] Q. Ma and S. de Kinderen, *Goal-Based Decision Making*. Cham: Springer International Publishing, 2016, pp. 19–35.

[56] C. M. Subramanian, A. Krishna, and A. Kaur, "Optimal goal programming of softgoals in goal-oriented requirements engineering," in *Proceeding of the 20th Pacific Asia Conference on Information Systems*, 2016.

[57] C. Subramanian, A. Krishna, R. Gopalan, and A. Kaur, "Quantitative reasoning of goal satisfaction in the i* framework." in *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering*, 2015.

[58] L. V. Kantorovich, "A new method of solving of some classes of extremal problems," *Dokl. Akad. Nauk SSSR*, vol. 28, no. 3, pp. 211–214, 1940.

[59] G. B. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," *Activity Analysis of Production and Allocation*, pp. 339–347, 1947.

[60] E. Beale and J. Forrest, "Global optimization using special ordered sets," *Mathematical Programming*, vol. 10, pp. 52–69, 1976.

[61] A. Affleck, A. Krishna, and N. R. Achuthan, "Non-functional requirements framework: A mathematical programming approach," *The Computer Journal*, vol. 58, pp. 1122–1139, 2015.

[62] A. Affleck, A. Krishna, and N. Achuthan, "Optimal selection of operationalizations for non-functional requirements," in *Proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling - Volume 143*, ser. APCCM '13. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2013, pp. 69–78.

[63] M. Feather and S. Cornford, "Quantitative risk-based requirements reasoning," *Requirements Engineering*, vol. 8, pp. 248–265, 2003.

[64] lp solve, "Introduction to lp_solve 5.5.2.0," http://lpsolve.sourceforge.net, April 2013.

[65] J. L. Bentley and R. Sedgewick, "Fast algorithms for sorting and searching strings," in *SODA*, vol. 97, 1997, pp. 360–369.

[66] G. H. Hardy, *A Mathematician's Apology*. Cambridge University Press, 1992.

.

Appendix A

GRL Constraint Proof

This appendix details the test graphs used to verify the initial GRL optimisation schema as detailed in chapter 4. Section 4.3 introduced this topic by demonstrating the behaviour of individual constraints. The results presented in this appendix used similar methods to determine the expected objective value function for each graph, after the test data was created the implemented simulation was used to generate actual results.

A.1 Decomposition Relationships

Section 4.3.1 and section 4.3.2 demonstrated the decomposition constraints when a single relationship was used. This section provides the decomposition test graphs, and their expected and actual objective function values when multiple AND, OR, and means-end relationships are involved. The graphs also contain a combination of nodes that both allow and dis-allow the optimisation solver to assigned leaf values ie: task/resource nodes versus goal/softgoal nodes.

Table A.1 shows several test graphs that use a decomposition and means-end combination, with two weighted root nodes and a shared child node. Table A.2 then presents several graphs demonstrating the interplay of multiple AND and OR relationships. Finally, table A.3 shows the effects of multiple shared children.

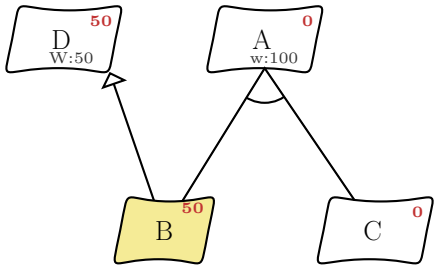
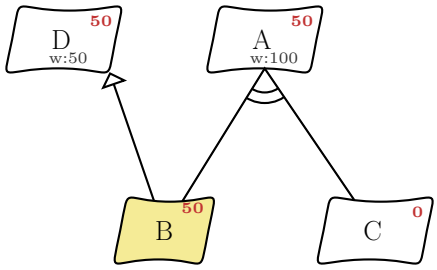
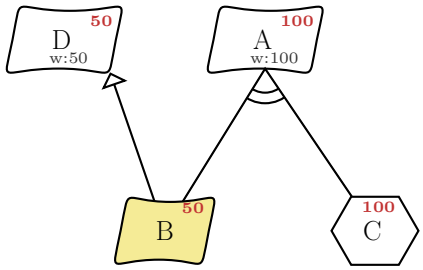
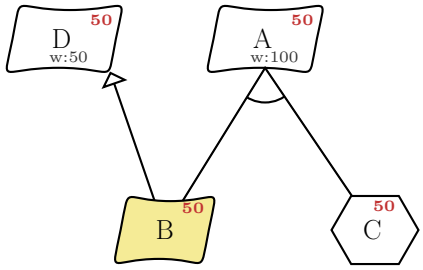
Graph	Objective Function
	Expected: 25 Actual: 25
	Expected: 75 Actual: 75
	Expected: 125 Actual: 125
	Expected: 75 Actual: 75

Table A.1: Decomposition graphs and their objective function values — single decomposition with means-end.

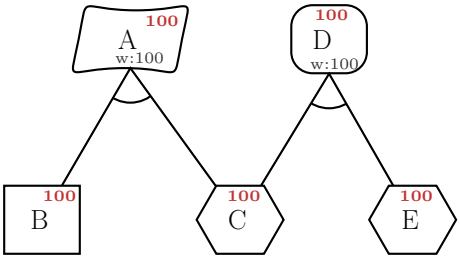
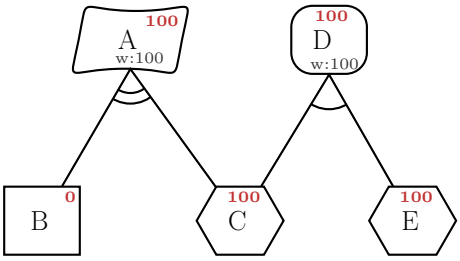
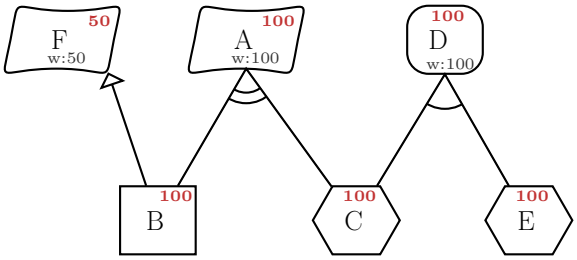
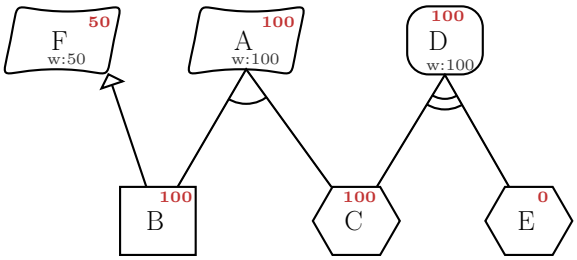
Graph	Objective Function
	Expected: 200 Actual: 200
	Expected: 200 Actual: 200
	Expected: 250 Actual: 250
	Expected: 250 Actual: 250

Table A.2: Decomposition graphs and their objective function values — multiple decompositions, and means-end.

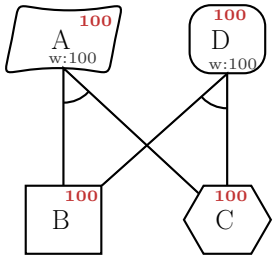
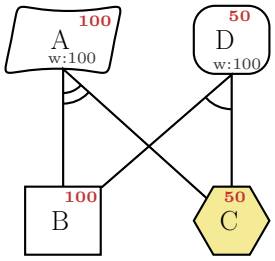
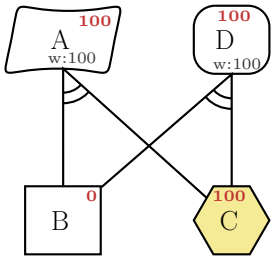
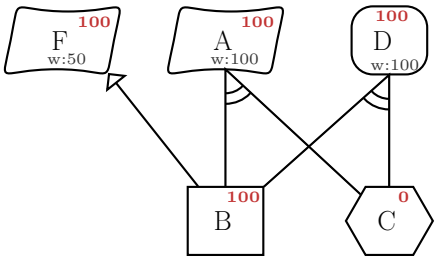
Graph	Objective Function
	Expected: 200 Actual: 200
	Expected: 150 Actual: 150
	Expected: 200 Actual: 200
	Expected: 250 Actual: 250

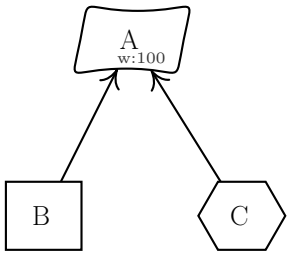
Table A.3: Decomposition graphs and their objective function values — multiple decomposition, with shared children.

A.2 Contribution Relationships

Section 4.3.3 elucidated how the contribution and correlation constraints force nodes to take the correct value, but the demonstration only used two contribution values. Figure A.1 presents a graph with similar structure, but with a range of values, and a combination of the tolerance and capped value options. Figure A.2 introduces a second root node allowing the verification of opposing relationships. Figure A.3 introduces a combination of decomposition and correlation relationships, in order to fully test the effect of the contribution relationships, the two decomposition children nodes are given initialisation values. Figure A.4 adds a second root node to the relationship combination, introducing the trade-off of an opposing contribution, figure A.5 repeats these tests but swaps the root node weight, finally, figure A.6 changes the decomposition relationship to an OR relation. A comparison of the results presented in the last three figures elucidates how these minor changes effect the values of nodes and the final objective function score.

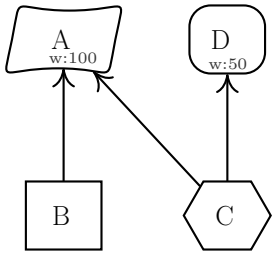
A.3 Dependency Relationships

Section 4.3.4 did not demonstrate individual constraints due to their similarity to those used in decomposition. The section shows several graph structures with varying input data in order to verify those constraints indeed function as intended. Figure A.7 shows a single dependency relationship with values ranging from -100 to 100. Figure A.8 then demonstrates multiple dependency relationships in order to verify that the root node does indeed take the minimum value. Finally, figure A.9 adds a decomposition relationship in order to ascertain that the dependency does override a pre-existing value if appropriate, and allow it to remain unchanged if possible.



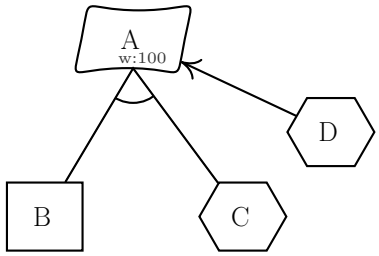
Input Values				Objective Function		Results		
(A,B)	**(A,C)**	**Tol**	**Cap**	**Expected**	**Actual**	**A**	**B**	**C**
90	50			100	100	100	100	20
90	50	X		90	90	90	100	0
100	50	X		100	100	100	100	0
90	50		X	100	100	100	100	20
90	50	X	X	90	90	90	100	0
100	50	X	X	100	100	100	100	0
90	-50			100	100	100	100	-20
90	-100	X		90	90	90	100	0
100	-50	X		100	100	100	100	0
90	-50		X	90	90	90	100	0
90	-50	X	X	90	90	90	100	0
100	-50	X	X	100	100	100	100	0

Figure A.1: Contribution and correlation test data — single parent, two children.



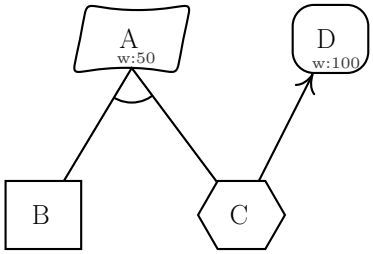
Input Values					Objective Function		Results			
(A,B)	**(A,C)**	**(D,C)**	**Tol**	**Cap**	**Expected**	**Actual**	**A**	**B**	**C**	**D**
80	60	100			150	150	100	50	100	100
80	60	100	X		140	140	90	37.5	100	100
80	-20	50			85	85	60	100	100	50
80	-60	50			80	80	80	100	0	0
20	-60	40			60	60	80	100	-100	-40
20	-60	40		X	40	40	20	100	0	0
-40	76	-42			83.42	83.42	100	-100	78.94	-33.16
-40	76	-42		X	55	55	76	0	100	-42
76	23	-42			78	78	99	100	100	-42
76	23	-42		X	78	78	99	100	100	-42
25	25	-50			25	25	25	100	0	0
25	25	-50		X	25	25	25	100	0	0

Figure A.2: Contribution and correlation test data — two parents, one shared child, one solo child.



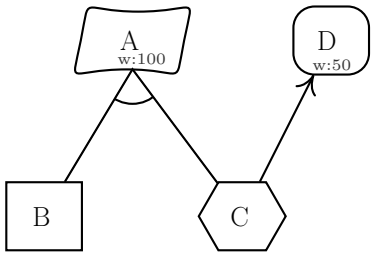
Input Values					Objective Function		Results	
B	**C**	**(A,D)**	**Tol**	**Cap**	**Expected**	**Actual**	**A**	**D**
50	50	80			100	100	100	62.5
50	50	80	X		90	90	90	50
50	50	100			100	100	100	50
100	100	40	X		100	100	100	0
50	50	-30			80	80	80	-100
50	50	-30		X	50	50	50	0

Figure A.3: Contribution and correlation test data — single parent with both a contribution relationship and a decomposition relationship.



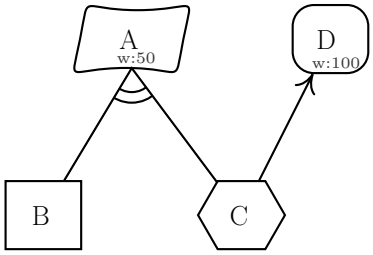
Input Values			Objective Function		Results			
(D,C)	**Tol**	**Cap**	**Expected**	**Actual**	**A**	**D**	**B**	**C**
-100			50	50	-100	0	-100	100
-100		X	0	0	0	0	0	0
-20			30	30	100	100	100	-20
-20		X	30	30	100	100	100	-20
40			90	90	100	100	100	40
40	X		90	90	100	100	100	100

Figure A.4: Contribution and correlation test data — one parent with a decomposition relationship, one parent with a contribution, one shared child node.



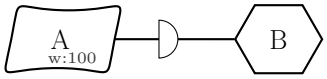
Input Values			Objective Function		Results			
(D,C)	**Tol**	**Cap**	**Expected**	**Actual**	**A**	**D**	**B**	**C**
-100			50	50	100	100	100	-100
-100		X	50	50	100	100	100	-100
-20			90	90	100	100	100	-20
-20		X	90	90	100	100	100	-20
40			120	120	100	100	100	40
40	X		120	120	100	100	100	40

Figure A.5: Contribution and correlation test data — same data as figure A.4, but with flipped node weights.



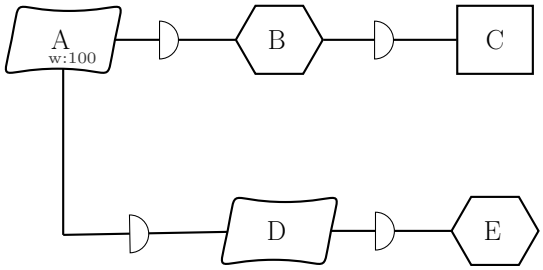
Input Values (D,C)	Tol	Cap	Objective Function Expected	Actual	Results A	D	B	C
-100			150	150	100	100	-100	100
-100		X	50	50	100	100	0	0
-20			70	70	100	100	-100	20
-20		X	50	50	100	100	0	0
40			90	90	100	0	100	40
40	X		90	90	100	0	100	40

Figure A.6: Contribution and correlation test data — same data as figure A.4, but with an OR relationship.



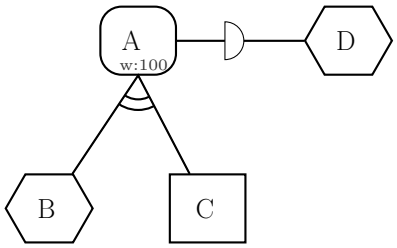
Input Values B	Objective Function Expected	Actual
50	50	50
100	100	100
90	90	90
-50	-50	-50
-90	-90	-90
-100	-100	-100
0	0	0

Figure A.7: Dependency Test Data — single dependency relationship



Input Values		Objective Function		Results		
C	**E**	**Expected**	**Actual**	**A**	**B**	**D**
-	-	100	100	100	100	100
50	-50	-50	-50	-50	50	-50
100	50	50	50	50	100	50
-100	100	-100	-100	-100	-100	100
20	0	0	0	0	20	0
-20	0	-20	-20	-20	-20	0

Figure A.8: Dependency test data — multiple dependency relationships



Input Values			Objective Function		Results
B	**C**	**D**	**Expected**	**Actual**	**A**
100	70	100	100	100	100
20	70	-50	-50	-50	-50
-20	-40	50	-20	-20	-20
-20	-40	-20	-20	-20	-20
40	70	70	70	70	70

Figure A.9: Dependency test data — dependency and decomposition relationships