Department of Mathematics and Statistics

Global Algorithms for Nonlinear Discrete Optimization and Discrete-Valued Optimal Control Problems

Siew Fang Woon

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

December 2009

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.

Siew Fang Woon

December 2009

Acknowledgments

This thesis would not have become a reality without the invaluable discussions I have had with several individuals, all of whom were overgenerous with their time. Foremost, I would like to express my deepest gratitude to my main supervisor, Volker Rehbock. Not only he has been a supportive supervisor, but most importantly a model of a true scholar. I admire and owe him for his broad knowledge and deep insight. This thesis is the result of our collaboration and I am grateful for his high commitment to quality and insistence on perfection. Such an exemple will serve as a guide throughout my career. I owe a deep appreciation to my co-supervisor, Kok Lay Teo, for creating opportunities for me to learn and grow as a researcher. Despite his tight schedule as the Head of Department, he always offered valuable advice and assistance throughout. My special thanks also go to Yong Hong Wu for his friendly assistance on Ph.D. related activities and Les Jennings from University of Western Australia for his kind technical support on MISER3.3. Further, I am particularly indebted to another great mentor, David Packer. He was always there to listen and offer sound advice on both technical and professional matters. I truly enjoyed our deep discussions. He has taught me so much that I will benefit from it throughout my life. I own him my eternal gratitude for proofreading parts of the thesis.

I am particularly indebted to my two fellow colleagues who have spent an enormous amount of precious time with me working through many challenges: thanks to Chun-Min Liu for his constant guidance on my study and other aspects of life; Ryan Loxton for sharing his optimal control and programming expertise,

invaluable insights and comments about my research. There is no overstating the importance of their help. My appreciations also go to Changzhi Wu for his generous technical discussions on optimal control; Greg Gamble for his assistance on LaTeX; Mark Grigoleit for his support on FORTRAN. Also my big thanks to the staff of the Department of Mathematics and Statistics for their administrative assistance throughout my study, especially Joyce, Lisa, Shuie, Florence, and Carey.

I wish to extend my heartfelt gratitude to my friends and colleagues, especially Khajee, Sarah, Zhouyu, Tiffany, Anamai, Ian, Wannika, Rose, Somkid, Richard, Sie Long, David, and Sammy, who have mentally and emotionally supported me throughout my study, and made me feel like I had a portion of a real life indeed. I would like to thank my fellow officemates for providing a friendly environment for my study. Further, it is my privilege to know the Ito, Cornthwaite, and Trinh families, who made my stay in Perth comfortable like at home. Besides, I am lucky to have received inspiration and encouragement from colleagues in my home country, especially Chooi Leng, Engku Muhammad Nazri, Hock Eam, and Lee Luan. Thanks for their support and friendship.

Last but not least, I would like to express my utmost appreciation to my entire family: my grandparents, parents, parents-in-law, aunties and uncles, sister-in-law, and cousins. They have always been there despite the ravages of time and distance. I would have never completed this thesis without their endless love and encouragement. To my wonderful partner Weng Hong, who has always been wholeheartedly supporting me overcoming countless difficulties in my life, no words can describe how lucky I am to have his unconditional support and love. For that I dedicate this thesis to him.

To everyone, I hope our paths continue to cross through the years and bless for the best.

Abstract

Optimal control problems arise in many applications, such as in economics, finance, process engineering, and robotics. Some optimal control problems involve a control which takes values from a discrete set. These problems are known as discrete-valued optimal control problems. Most practical discrete-valued optimal control problems have multiple local minima and thus require global optimization methods to generate practically useful solutions. Due to the high complexity of these problems, metaheuristic based global optimization techniques are usually required.

One of the more recent global optimization tools in the area of discrete optimization is known as the discrete filled function method. The basic idea of the discrete filled function method is as follows. We choose an initial point and then perform a local search to find an initial local minimizer. Then, we construct an auxiliary function, called a discrete filled function, at this local minimizer. By minimizing the filled function, either an improved local minimizer is found or one of the vertices of the constraint set is reached. Otherwise, the parameters of the filled function are adjusted. This process is repeated until no better local minimizer of the corresponding filled function is found. The final local minimizer is then taken as an approximation of the global minimizer.

While the main aim of this thesis is to present a new computational method for solving discrete-valued optimal control problems, the initial focus is on solving purely discrete optimization problems. We identify several discrete filled functions techniques in the literature and perform a critical review including comprehensive

numerical tests. Once the best filled function method is identified, we propose and test several variations of the method with numerical examples.

We then consider the task of determining near globally optimal solutions of discrete-valued optimal control problems. The main difficulty in solving the discrete-valued optimal control problems is that the control restraint set is discrete and hence not convex. Conventional computational optimal control techniques are designed for problems in which the control takes values in a connected set, such as an interval, and thus they cannot solve the problem directly. Furthermore, variable switching times are known to cause problems in the implementation of any numerical algorithm due to the variable location of discontinuities in the dynamics. Therefore, such problem cannot be solved using conventional computational approaches. We propose a time scaling transformation to overcome this difficulty, where a new discrete variable representing the switching sequence and a new variable controlling the switching times are introduced. The transformation results in an equivalent mixed discrete optimization problem. The transformed problem is then decomposed into a bi-level optimization problem, which is solved using a combination of an efficient discrete filled function method identified earlier and a computational optimal control technique based on the concept of control parameterization.

To demonstrate the applicability of the proposed method, we solve two complex applied engineering problems involving a hybrid power system and a sensor scheduling task, respectively. Computational results indicate that this method is robust, reliable, and efficient. It can successfully identify a near-global solution for these complex applied optimization problems, despite the demonstrated presence of multiple local optima. In addition, we also compare the results obtained with other methods in the literature. Numerical results confirm that the proposed method yields significant improvements over those obtained by other methods.

List of Publications Related to this Thesis

(1) S. F. Woon, V. Rehbock, and A. A. Setiawan. Modeling a PV-Diesel-Battery Power System: An Optimal Control Approach. In *Proceeding of International Conference of Modeling, Simulation and Control, World Congress of Engineering and Computer Science*, Berkeley, CA, October 22-24, 2008, pages 864–879.

(2) S. F. Woon, V. Rehbock, and R. C. Loxton. Global Optimization Method for Continuous-Time Sensor Scheduling. *Special Issue of Nonlinear Dynamics & Systems Theory*. Accepted.

(3) S. F. Woon and V. Rehbock. A Critical Review of Discrete Filled Function Methods in Solving Nonlinear Discrete Optimization Problems. Submitted.

(4) S. F. Woon, V. Rehbock, and R. C. Loxton. Optimal Operation of a Hybrid Power System. Submitted.

List of Conference Presentations Related to this Thesis

(1) S. F. Woon, V. Rehbock, and R. C. Loxton. Global Optimization in Hybrid Power System. *The Institute for Operations Research and the Management Sciences Annual Meeting*, Washington, DC, October 12-15, 2008.

(2) S. F. Woon, V. Rehbock, and A. A. Setiawan. Modeling a PV-Diesel-Battery Power System: An Optimal Control Approach. *International Conference of Modeling, Simulation and Control, World Congress of Engineering and Computer Science*, Berkeley, CA, October 22-24, 2008.

(3) S. F. Woon, V. Rehbock, and R. C. Loxton. Optimizing Hybrid Power System with Discrete Filled Function Method. *The* 1^{st} *Pacific Rim Mathematical Association Congress*, Sydney, NSW, July 5-10, 2009.

(4) S. F. Woon and V. Rehbock. Solving Global Optimization Problems with Discrete Filled Function Methods: A Survey. *The* 20^{th} *Meeting of the International Symposium of Mathematical Programming*, Chicago, IL, August 23-28, 2009.

(5) S. F. Woon, V. Rehbock, and R. C. Loxton. Global Discrete Valued Optimal Control. *The* 53^{rd} *Annual Meeting of the Australian Mathematical Society*, Adelaide, SA, September 28 - October 1, 2009.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Optimal control describes the task of determining feasible control policies for a given dynamical system to achieve a certain optimality criterion. Specifically, an objective functional is to be minimized subject to a dynamical system governing the behavior of the state variables and subject to constraints. Usually, the dynamical system constitutes a set of ordinary or partial different equations.

Optimal control problems can be found in many applications, such as economics, finance, engineering, and robotics. The first order necessary conditions of for an optimal control are described by the Euler-Lagrange equations. However, the Euler-Lagrange equations do not apply in the presence of bounds on the control. Instead, the first order necessary conditions of optimality for such problems can be determined by the minimum principle [110]. Application of the minimum principle developed by Pontryagin and his collaborators can solve many idealized problems and has found wide application in the theory of economics [50]. Necessary conditions of optimality are essentially stated in the forms of a two-point boundary value problem. When this cannot be solved analytically, as is often the case, one must resort to numerical methods. One of the common techniques for solving two-point boundary value problems is the multiple shooting method. This method divides the time horizon into several subintervals, solves an initial value problem over each of these intervals, and imposes additional matching conditions to form a complete

solution of the problem [18, 103, 109].

Another well-known principle, known as dynamic programming principle [6], is introduced by Bellman to solve optimal control problems. In contrast to the minimum principle, the technique often yields the optimal control in a feedback form. While many optimal control problems have been solved by this technique, the application of the dynamic programming principal requires the solution of Hamilton-Jacobi-Bellman (HJB) partial differential equation, which then yields the optimal value function from which the optimal control may be determined. For most practically significant problems, though, it is difficult to solve the Hamilton-Jacobi-Bellman equation directly. Many applications based complex optimal control problems cannot be solved analytically by any of the means discussed above. Even the numerical solution of the HJB is usually difficult to determine, especially in the case of high dimensional problems.

Since the introduction of computers in 1950s, many computational procedures have been developed to solve complex optimal control problems numerically. Over the years, numerical solution techniques have solved a broad range of practical optimal control problems successfully. Following the publication of [145], new interest has emerged in the area of optimal control computation. Recent trends in the area are:

- The recognition that many practical problems involve impulsive or hybrid systems [22, 126, 157];
- The need to solve practical problems where the control takes values from a discrete set [46, 56, 111, 134];
- The practical need to determine a globally rather than just locally optimal solution [11];
- The ability to test sufficient conditions for optimality numerically [92, 93, 94].

In this thesis, we focus on optimal control problems involving controls which take values from discrete sets. Such problems are known as discrete-valued optimal

control problems. Solving the discrete-valued optimal control problems has been a challenging task since the control restraint set is discrete and hence not convex. Furthermore, many practical discrete-valued optimal control problems have more than one locally optimal solution, thus leading to the challenge of determining the best solution amongst these multiple local optima. The best solution obtained is known as the global solution. In the next section, we review several computational methods for solving generic optimal control problems. Then, we briefly discuss the special nature of discrete-valued optimal control problems followed with a discussion of global optimization methods.

1.2 Computational Methods for Solving Optimal Control Problems

Over the years, many computational methods have been developed to solve a broad class of complex optimal control problems. All methods involve a partition of the time horizon and many require a discretized approximation of the control in some form. Some methods also discretize the state of the problem and therefore the differential equations describing the system dynamics. Most methods ultimately arrive at an approximating mathematical programming problem which, in turn, can be solved by a variety of optimization techniques. In this section, we look at several of the more popular numerical solution techniques for optimal control problems.

1.2.1 Direct Collocation (DIRCOL)

A special transcription method, DIRCOL converts a constrained optimal control problem into a finite dimensional nonlinear constrained optimization problem by an appropriate discretization of both control and state variables. The transformed problem, the dimension of which depends on the discretization grid, can be solved by standard quadratic programming (SQP) methods [28, 118, 119]. A detail description of the method is given in [148, 149]. Although it can readily solve small scale problems, the discretization of both control and state variables for large scale

problems requires excessive computational time and storage.

1.2.2 Dynamic Programming and Iterative Dynamic Programming (IDP)

Bellman's principle of optimality has made a significant contribution in a wide range of applications in optimal control. Some extensions and variation of the principle are discussed in [47, 72, 73, 74, 75, 86]. The IDP technique is loosely based on Bellman's principle of optimality. The method uses a grid structure for discretizing both the state variables and the controls. Accessible points in the state trajectory and admissible control values are defined on grids constructed in the state and control space, respectively. The grids are refined iteratively until a satisfactory control policy is obtained. The technique was initially developed in [74] and then refined in [72, 78] to improve the computational efficiency. The early version of the method used piecewise constant controls and this was later extended to piecewise linear continuous control policies [75]. Constraints are incorporated into the objective function using a penalty function method. It has found widespread application in the area of chemical engineering [76, 78, 79, 82, 117, 135]. However, the method can be difficult to use due to the presence of many user defined parameters driving the algorithm.

1.2.3 Luus-Jaakola (LJ) Optimization Procedure

The LJ optimization procedure [85] is a direct search optimization technique based on randomly chosen points and an adaptive reduction of the search space. An initial control estimator is taken at each of a predetermined number of stages along with an initial region size for the control used at each stage. In each iteration, a set of control values is generated randomly and is used to evaluate an augmented objective function. The best of these over a predetermined number of iterations is then taken as the solution of that stage. The search region is contracted by a chosen region contraction factor. This process is repeated until a specified number of passes is reached or the convergence criterion is satisfied. The LJ optimization procedure has

been successfully applied to solve a broad class of practical problems, such as those in [77, 80, 81, 83, 84]. A major problem with the method is the large number of function evaluations required. The method also requires careful tuning making it more suitable for expert users.

1.2.4 Control Parameterization

In control parameterization, the time horizon of an optimal control problem is partitioned into several subintervals such that each control can be approximated by a piecewise polynomial function consistent with the corresponding partition. Often, the control function is expressed as a linear combination of a polynomial spline where the coefficients of the function determine the control. These coefficients are known as the control parameters. The more intervals used in a partition, the more accurate are the solution it yields. As a results, an optimal control problem becomes a finite dimensional optimal parameter selection problem, which is essentially a mathematical programming problem. Thus, the solution of the resulting problem can be readily obtained by existing optimization software packages, such as NPSOL [33], NLPQL [119], FFSQP [164], based on the sequential quadratic programming method. A convergence analysis of this approach can be found in [130, 131, 132, 133, 140]. In addition, [121] discusses the appropriate choice of both the control parameterization and the numerical solution scheme for the underlying dynamical system. Calculation of the gradients with respect to the control parameters needs to be preformed in a roundabout manner and usually requires the solution of a set of costate differential equations [140]. Some applications of the control parametrization technique can be found in [9, 35, 36, 42, 48, 49, 130, 131, 132, 133, 136, 137, 138, 139, 140, 141, 142, 143, 144, 152].

1.2.5 Recursive Integration Optimal Trajectory Solver (RIOTS)

RIOTS is one of the toolbox designed for Matlab [123] for solving optimal control problems. The basic idea behind RIOTS is to approximate controls by finite-dimensional B-splines, which is an example of the control parameterization ap-

proach. The integration of the system dynamics is carried out using fixed step-size Runge-Kutta integration. A detailed description of the implementation of RIOTS in the application examples can be found in [100, 122, 123]. However, RIOTS has some limitations on type of problems it can solve effectively [123]. For instance, it has difficulty in solving problems involve inequality state constraints which require a high level of discretization; the computation of gradients for path constraints are not handled as efficiently as expected; and the selection of the control subspaces affects both the accuracy of numerical integration and the approximate solutions to the original problem.

1.2.6 Sequential Gradient Restoration Algorithms

While this approach is also based on control parameterization, the underlying mathematical programming problem is not solved in the usual manner. Instead, its solution involves a sequences of two phase cycles: a gradient phase and a restoration phase. The first phase minimizes the augmented objective function while the latter reduces the resulting constraint violation. A detailed description and analysis of the method can be found in [96, 97, 98, 99].

1.2.7 Leap-Frog Algorithm

Initially developed in [108], the leap-frog algorithm is used to solve a special types of two point boundary value problems. The algorithm is further developed in [53, 54] to handle general nonlinear systems with unbounded and bounded controls. A piecewise optimal trajectory is obtained in each subinterval, where the junctions of these sub-trajectories are updated through a scheme of midpoint maps. A thorough description of the algorithm is outlined in [53, 54] to solve a class of optimal control problems with bounded controls in the plane.

1.2.8 Switching Time Computation (STC) Method

The STC is a computational procedure developed in [52] to determine suitable places of switchings for single-input nonlinear systems. A concatenation of constant-

input arcs is applied to solve the dynamics from a given initial point to the target. The gradients with respect to the switching times variables are computed without the use of costate equations. The method is incorporated in a time optimal bang-bang (TOBC) control algorithm in [51, 71]. Although the STC method can be fast compared with other optimal control software, there is a limited class of problems which can be solved by this method. For instances, many types of constraints cannot be handled directly without using penalty methods.

1.2.9 MISER3.3

MISER3.3 [48] is an optimal control software package based on the control parameterization technique as described above. It can handle several types of constraints in solving optimal control problems, including all time inequality constraints on the state. The package is designed to deal with a general canonical form of optimal control problems, thus making it widely applicable. MISER has been successfully applied to solve many practical optimal control problems [11, 21, 48, 59, 61, 62, 60, 63, 64, 66, 67, 68, 111, 140, 142, 143, 146].

1.3 Discrete-Valued Optimal Control Problems

In discrete-valued optimal control problems, the control is restricted to a set of discrete values. Examples include the design of operating procedures of a chemical plant (start up, shut down, and changeovers) [56], management of batteries in a submarine [111], optimal driving strategies for a train [46], the submarine transit path problem [11], and switched amplifier design [134]. To solve such problems, it is necessary to find the optimal sequence of discrete control values and the optimal switching times between changing control actions. Solving discrete-valued optimal control problems involves several additional challenges which as not encountered with continuous valued optimal control problems. These include:

- The feasible region of the underlying mathemtical programming problem is discrete and hence not convex.

- The gradients of the objective and constraint functions with respect to the switching times are not differentiable [140], thus making it difficult to implement efficient gradient based optimization methods to calculate exact values for the switching times.

- Numerical integration of the dynamics become difficult when variable switching times are involved, as the knot points in the integration scheme require updating whenever the switching times change [70].

- The number of possible switching sequences is extremely large in many examples. In fact, finding an optimal switching sequence is a combinatorial optimization problem which is well known to be difficult to solve [64].

A variety of approaches to solve discrete-valued optimal control problems in the literature is reviewed in [129]. This include the control parameterization enhancing technique (CPET) [64], stochastic methods [13, 114, 158], standard methods using excessive refinement [79], and exact semi analytical methods [55]. Among these methods, CPET appears to be the only effective numerical technique which is widely applicable to this class of optimal control problems [129]. CPET transforms a discrete-valued optimal control problem into an equivalent optimal parameter selection problem. Under this transformation, the switching points are mapped onto a set of fixed knots in the new time scale, and the transformed problem is an ordinary optimal control problem with known and fixed switching instants. Hence, such problems can be readily solved by many existing optimal control techniques such as control parameterization. The effectiveness of CPET in determining exact switching instants of a control policy has been proven in [59, 60, 61, 62, 63, 143] for various nontrivial problems.

Although CPET overcomes the first three difficulties mentioned above, it does not handle the last issue effectively as it introduces many artificial switchings in order to capture more possible orderings of the sequence of discrete control values when solving the transformed problem. Furthermore, the CPET approach is generally not able to determine a global or near global optimal switching sequence. With

the addition of a large number of artificial switches, the resulting optimization problem has many more local minima and many of these have relatively high objective values. Note that many practical discrete-valued optimal control problems exhibit similar behavior, thus making it harder to determine the global optimal solution of such problems.

1.4 Global Optimization

Most practical discrete and mixed discrete optimization problems are nonlinear and known to have more than one locally optimal solution. This suggests the need for global optimization techniques which seek the best solution amongst multiple local optima. Global optimization problems may be unconstrained or constrained, and different algorithms have been developed, depending on whether constraints are present as well as on the nature of these constraints.

The challenge in global optimization is to avoid being trapped in the basins surrounding local minimizers. Several global methods have been proposed for solving discrete optimization problems. These techniques can be classified into two main categories: exact methods and metaheuristic methods. The branch and bound method [41, 65, 87], the cutting plane method [23, 29, 163], Lagrangian relaxation [27, 32], the nonlinear Lagrangian relaxation method [157, 159], the discrete Lagrangian methods [155, 156], dynamic programming [90], and relaxation techniques [15, 40, 113] are popular exact methods. These exact methods can ensure that a global solution is found when solving small size discrete optimization problems. However, such methods require excessive computational time when solving large scale problems. Furthermore, only well-structured problems with good analytical properties can be solved efficiently using these exact methods.

Since nonlinear discrete optimization problems are generally NP-hard, there are no exact algorithms with polynomial-time complexity for solving them. Hence, a metaheuristic computational approach is required, especially for high-dimensional problems. The term metaheuristic is derived from two Greek words, where *heuristic*

(heuriskein) means *to find* while the suffix *meta* refers to *beyond, in an upper level* [8]. A heuristic is a technique to find a good feasible solution where the global optimality is not crucial. Often, a heuristic method is problem-specific and designed to obtain conceptual simplicity [8, 10, 43]. A metaheuristic is a higher level heuristic algorithm for solving a general class of optimization problems. It is a master strategy which combines different methods for exploring search space efficiently in determining a near-optima solution. Thus, metaheuristics often produce higher quality results than classical heuristics though they generally require longer computational times. In addition, metaheuristics are capable of solving a variety of complex application problems and they avoid getting trapped in basins associated with local extreme points. The metaheuristic methods include greedy-search [3, 17, 20, 24], simulated annealing [101, 115], genetic algorithms [12, 147, 154], tabu search [34, 95], and filled function techniques. Though these methods cannot guarantee a global solution, satisfactory results can often be found for high dimensional nonlinear discrete optimization problems in a reasonable amount of computational time.

1.5 Discrete Filled Function Method

The discrete filled function method is one of the more recently developed global optimization tools for discrete optimization problems. Once a local minimum has been determined by an ordinary descent method, the discrete filled function approach introduces an auxiliary function to avoid entrapment in the basin associated with this minimum. The local minimizer of the original function becomes a local maximizer of the auxiliary function. By minimizing the auxiliary function, the search moves away from the current local minimizer in the hope of escaping the basin associated with this minimizer. Note that the auxiliary function is defined in terms of one or more parameters and needs to possess certain properties, details of which are discussed in Chapter 2.

The first filled function was introduced by Ge in the late 1980s [30] in the context of solving continuous global optimization problems. In [31], Ge and Huang

extended the continuous filled function concept to solve nonlinear discrete optimization problems, where a continuous global optimization problem is formulated to approximate the discrete global optimization problem, before solving it by the continuous filled function method. When a global minimizer of the continuous approximation is found, the nearest integer point is used to approximate the global solution of the discrete problem. However, the approximating continuous optimization problem always generates more local minimizers than the original discrete one, thus making it more difficult to determine a global solution. Numerical results reported in [105] have shown that the true global minimizer is difficult to determine using this approach. A detailed analysis of the continuous filled function approach can also be found in [105].

Zhu [165] is believed to be the first researcher to introduce a true discrete equivalent of the continuous filled function method in late 1990s. Such an approach is now known as a discrete filled function method or discrete global descent method. A discrete filled function method is able to overcome the difficulties encountered in using a continuous approximation, as discussed above. However, the filled function proposed by Zhu contains an exponential term, which consequently makes it difficult to determine a point in a lower basin [105, 106]. Since then, several types of discrete filled functions with improved theoretical properties have been proposed in [38, 106, 107, 127, 128, 160, 161, 162] to enhance computational efficiency.

The discrete filled function approach can be described as follows. An initial point is chosen and a local search is applied to find an initial discrete local minimizer. Then, an auxiliary function, called a filled function, is constructed at this local minimizer. By minimizing the filled function, either an improved discrete local minimizer is found or the boundary of the feasible region is reached. The discrete local minimizer of the filled function usually becomes a new starting point for minimizing the original objective with the hope of finding an improved point compared to the first local minimizer. A new filled function is constructed at this improved point. The process is repeated until no improved local minimizer of the earlier filled function can be found. The final discrete local minimizer is then taken

as an approximation of the global minimizer.

If a local minimizer of the filled function cannot be found after repeated searches terminate on the boundary of the box constrained feasible region, the parameters defining the filled functions are adjusted and the search is repeated. This adjustment of the parameters continues until the parameters reach their predetermined bounds; the best solution obtained so far is then taken as the global minimizer. Note that some filled functions have one parameter (such as those in [38, 127, 161]), while the rest are equipped with two parameters. The latter filled functions often have one parameter which is partially dependent on the other and this requires additional steps when tuning the parameters in order to satisfy the required convergence criteria. Note that each filled function discussed here has unique characteristics. The complexity of each filled function is also dependent on its associated algorithm, as discussed in detail in the following chapter.

Filled function methods have been a popular global optimization tool in recent years. However, there has been limited attention on investigating this method in the context of mixed discrete optimization problems in particular the class of discrete valued optimal control problems.

1.6 Objectives

The main purpose of this research is to construct and test a global algorithm which incorporates a discrete filled function method into a computational optimal control algorithm capable of solving a general class of discrete-valued optimal control problems. To start with, we introduce a modified time scaling transformation that results in a transformed problem which has far fewer variables than that resulting from the standard CPET approach. The resulting problem has fewer local minimizers and a global solution can thus be obtained with less computational effort. It is essentially a mixed discrete optimization problem and the next stage of our proposed method involve its decomposition into a purely discrete optimization problem and an ordinary optimal parameter selection problem. We then apply an effective discrete filled

function algorithm which is able to bypass locally optimal solutions and thus yield a solution closer to the global one. A standard optimal control package, MISER3.3, is used to solve the subproblems which appear at every iteration of the discrete optimization. We apply our proposed algorithm to two discrete-valued optimal control problems in engineering: the hybrid power system problem and sensor scheduling problem. The first problem involves a hybrid system which requires an operating schedule to minimize the total operating cost of a PV(photovoltaic)-diesel-battery hybrid power system. The second problem involves the operation schedule of a set of sensors over a given time frame to reduce the overall signal estimation error. We attempt to determine the global optimal solution for both application problems which are well known to have multiple local minima. We summarize our research objectives as follows:

- To review existing discrete filled function methods and compare their computational efficiency when solving discrete optimization problems and mixed discrete optimization problems.
- To develop new discrete filled function algorithms to solve discrete optimization problems efficiently.
- To introduce a transformation where a discrete-valued optimal control problem is transformed into an equivalent mixed discrete optimization problem.
- To propose a decomposition of this problem into a discrete upper level and a continuous lower level problem.
- To apply an effective discrete filled function method to the upper level problem.
- To determine a near global solution of minimizing the operation cost of a hybrid power system.
- To determine a near global solution of minimizing the error estimation of a general sensor scheduling problem.

- To embed the discrete filled function algorithm into MISER3.3, the optimal control software, in such a way that a near globally optimal solution is obtained when solving discrete-valued optimal control problems.

1.7 Significance of the Study

A new algorithm based on the discrete filled function method and on a conventional computational optimal control algorithm is proposed to solve discrete-valued optimal control problems. Numerical results demonstrate that the method is able to bypass locally optimal solutions and thus yield a solution closer to the global one in solving two complex application problems. In addition, we review a range of discrete filled function methods and suggest some effective variations to improve their efficiency.

1.8 Thesis Overview

This thesis is divided into six chapters that are organized as follows. Chapter 2 reviews several discrete filled functions and their associated algorithms as proposed in the literature. Some basic discrete optimization concepts and a generic discrete filled function algorithm are presented. Then, several individual discrete filled function formulations, their properties, and particulars of their associated algorithms are also discussed. The performances of selected filled function algorithms when applied to several test problems are compared. The most promising filled function method is identified, and a various of modification of this method are proposed and tested in Chapter 3. Next, a new metaheuristic which incorporates the discrete filled function algorithm into a standard optimal control software is proposed for solving two applied discrete-valued optimal control problems.

Chapter 4 proposes a new algorithm for determining an operating schedule that minimizes the total operating cost of a PV-diesel-battery hybrid power system. The hybrid power system consisting of a diesel generator as the main component, with a PV array providing additional energy and a battery bank for storage. An

earlier model developed in [116] is considered and we demonstrated that this model has many local minimizers. The outcomes obtained from the proposed algorithm are compared with the results in the literature. Numerical results for different numbers of allowed switches are also presented in this chapter.

Chapter 5 discusses a general class of optimal sensor scheduling problems. The scheduling of an operation of sensors over a given time frame, where only one sensor may be active at any one time, is required to minimize the signal estimation error. The sensor problem is first formulated as a discrete-valued optimal control problem. Then, the problem is transformed into an equivalent mixed discrete optimization problem to determine its global solution. The proposed global optimization algorithm is applied to solve this problem. To evaluate the effectiveness of the proposed algorithm, the results are compared with those obtained in the literature at the end of this chapter.

Finally, Chapter 6 summarizes the findings of the study. Limitations of the study are discussed and possible directions for future research work are also suggested.

Chapter 2

A Review of Discrete Filled Function Methods

This chapter begins with some basic definitions and concepts used in the discrete optimization area, followed with a generic algorithm of discrete filled function. The 3-hump back camel function [19] is used to illustrate how the filled function algorithm works. Nine variations of the discrete filled function method in literature are identified and a review on theoretical properties of each method is discussed. The most promising filled functions are tested on several test problems. The performances of these selected filled function algorithms are compared at the end of this chapter.

2.1 Introduction

Many real life applications, such as production planning, finance, scheduling, and operations involve integer valued decision variables. We distinguish between discrete optimization problems, where all decision variables are integer valued, and mixed discrete optimization problems, where only some of the decision variables have integer values. The latter type are often decomposed into purely discrete and continuous subproblems, respectively, and hybrid algorithms for their solutions are developed on this basis. The discrete parts of these hybrid algorithms are similar in nature to the purely discrete algorithms, which we address in this chapter. Consider

the following nonlinear discrete optimization problem:

$$\min f(\mathbf{x}), \quad \text{s.t. } \mathbf{x} \in X, \tag{2.1}$$

where $X = \{\mathbf{x} \in \mathbb{Z}^n | \mathbf{x}_{i,\min} \leq \mathbf{x}_i \leq \mathbf{x}_{i,\max}\}$, $\mathbb{Z}^n$ is the set of integer points in $\mathbb{R}^n$, and $\mathbf{x}_{i,\min},\ \mathbf{x}_{i,\max},\ i = 1, \ldots, n$, are given bounds. Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be any two distinct points in the box constrained set X and make the following assumptions:

Assumption 2.1 *There exists a constant $\mathcal{K}$ satisfying*

$$1 \leq \max_{\substack{\mathbf{x}_1, \mathbf{x}_2 \in X \\ \mathbf{x}_1 \neq \mathbf{x}_2}} \| \mathbf{x}_1 - \mathbf{x}_2 \| \leq \mathcal{K} < \infty,$$

where $\| \cdot \|$ is the Euclidean norm.

Assumption 2.2 *There exists a constant $\mathcal{L}$, $0 < \mathcal{L} < \infty$, such that*

$$|f(\mathbf{x}_1) - f(\mathbf{x}_2)| \leq \mathcal{L} \| \mathbf{x}_1 - \mathbf{x}_2 \| .$$

Most discrete filled function methods are designed to solve box constrained problems. Unconstrained and more generally constrained problems may be converted into an equivalent box constrained form. For example, consider the following unconstrained discrete optimization problem,

$$\min f(\mathbf{x}), \quad \text{s.t. } \mathbf{x} \in \mathbb{Z}^n, \tag{2.2}$$

If f is coercive, i.e., $f(\mathbf{x}) \to +\infty$ as $\| \mathbf{x} \| \to +\infty$, then there exists a box which contains all discrete minimizers of f. Hence, the formulation in (2.2) can be transformed into an equivalent formulation in (2.1) and can thus be solved by any discrete filled function method. Many discrete filled function algorithms in the literature, such as [107, 106, 162, 38], are also directly applicable to linearly constrained problems as long as the resulting feasible region is convex and pathwise connected.

As for generally constrained problems, the nonlinear constraints are usually handled with a penalty method. Consider the following general nonlinear constrained discrete optimization problem,

$$\min g_0(\mathbf{x}), \quad \text{s.t. } \mathbf{x} \in \Lambda, \tag{2.3}$$

where $\Lambda = \{\mathbf{x} \in \mathbb{Z}^n : g_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, m\}$ and $\mathbb{Z}^n$ is a set of integer points in $\mathbb{R}^n$. In [105], the constrained problem (2.3) is converted into an equivalent box constrained problem by adding a penalty term to the objective function f, i.e.

$$f(\mathbf{x}) = g_0(\mathbf{x}) + \alpha_0 \sum_{i=1}^{m} \max\{0, g_i(\mathbf{x})\} \tag{2.4}$$

or

$$f(\mathbf{x}) = g_0(\mathbf{x}) + \alpha_0 \sum_{i=1}^{m} [\max\{0, g_i(\mathbf{x})\}]^2, \tag{2.5}$$

where α_0 is a sufficiently large parameter. Note that it is difficult to determine an exact penalty parameter when solving these NP-hard problems and thus only approximate solutions can be determined. Note also that the discrete filled function method in [161] takes a different approach and incorporates constraints directly into the formulation of the filled function.

2.2 Discrete Optimization: Concepts and Approach

2.2.1 Preliminary Concepts

We recall some relevant definitions and concepts used in the discrete optimization area.

Definition 2.1 *A sequence $\{\mathbf{x}^{(i)}\}_{i=0}^{k+1}$ between two distinct points $\mathbf{x}^*$ and $\mathbf{x}^{**}$ in X is a discrete path in X if $\mathbf{x}^{(0)} = \mathbf{x}^*,\ \mathbf{x}^{(k+1)} = \mathbf{x}^{**},\ \mathbf{x}^{(i)} \in X$ for all i, $\mathbf{x}^{(i)} \neq \mathbf{x}^{(j)}$ for $i \neq j$, and $\| \mathbf{x}^{(i+1)} - \mathbf{x}^{(i)} \| = 1$ for all i. If such a discrete path exists, then $\mathbf{x}^*$ and $\mathbf{x}^{**}$ are pathwise connected in X. If every two distinct points in X are pathwise connected in X, then X is a pathwise connected set.*

Definition 2.2 *For any $\mathbf{x} \in X$, the neighbourhood of $\mathbf{x}$ is defined by $N(\mathbf{x}) = \{\mathbf{w} \in X | \ \mathbf{w} = \mathbf{x} \pm \boldsymbol{e}_i : i = 1, 2, \ldots, n\}$. Here, $\boldsymbol{e}_i$ denotes the i-th standard unit basis vector of $\mathbb{R}^n$, with the i-th component equal to one and all other components equal to zero.*

Definition 2.3 *The set of all feasible directions at $\mathbf{x} \in X$ is defined by $\mathcal{D}(\mathbf{x}) = \{\mathbf{d} \in \mathbb{Z}^n : \mathbf{x} + \mathbf{d} \in N(\mathbf{x})\} \subset E = \{\pm \boldsymbol{e}_1, \ldots, \pm \boldsymbol{e}_n\}$.*

Definition 2.4 $\mathbf{d} \in \mathcal{D}(\mathbf{x})$ *is a descent direction of* f *at* $\mathbf{x}$ *if* $f(\mathbf{x}+\mathbf{d}) < f(\mathbf{x})$.

Definition 2.5 $\mathbf{d}^* \in \mathcal{D}(\mathbf{x})$ *is a discrete steepest descent direction of* f *at* $\mathbf{x}$ *if it is a descent direction and* $f(\mathbf{x}+\mathbf{d}^*) \leq f(\mathbf{x}+\mathbf{d})$ *for any* $\mathbf{d} \in \mathcal{D}(\mathbf{x})$.

Definition 2.6 $\mathbf{x}^* \in X$ *is a local minimizer of* X *if* $f(\mathbf{x}^*) \leq f(\mathbf{x})$ *for all* $\mathbf{x} \in N(\mathbf{x}^*)$. *If* $f(\mathbf{x}^*) < f(\mathbf{x})$ *for all* $\mathbf{x} \in N(\mathbf{x}^*)$, *then* $\mathbf{x}^*$ *is a strict local minimizer of* f.

Definition 2.7 $\mathbf{x}^*$ *is a global minimizer of* f *if* $f(\mathbf{x}^*) \leq f(\mathbf{x})$ *for all* $\mathbf{x} \in X$. *If* $f(\mathbf{x}^*) < f(\mathbf{x})$ *for all* $\mathbf{x} \in X \setminus \mathbf{x}^*$, *then* $\mathbf{x}^*$ *is a strict global minimizer of* f.

Definition 2.8 $\mathbf{x}$ *is a vertex of* X *if, for each* $\mathbf{d} \in \mathcal{D}(\mathbf{x})$, $\mathbf{x}+\mathbf{d} \in X$ *and* $\mathbf{x}-\mathbf{d} \notin X$. *Let* $\tilde{X}$ *denote the set of vertices of* X.

Definition 2.9 $B^* \subset X$ *is a discrete basin of* f *corresponding to* $\mathbf{x}^*$ *if it satisfies the following conditions:*

- *It is pathwise connected;*
- *It contains* $\mathbf{x}^*$*;*
- *For each* $\mathbf{x} \in B^*$, *any connected path consisting of descent steps and starting at* $\mathbf{x}$ *converges to* $\mathbf{x}^*$.

Definition 2.10 *Let* $\mathbf{x}^*$ *and* $\mathbf{x}^{**}$ *be two distinct local minimizers of* f. *If* $f(\mathbf{x}^{**}) < f(\mathbf{x}^*)$, *then the discrete basin* B^{**} *of* f *associated with* $\mathbf{x}^{**}$ *is said to be lower than the discrete basin* B^* *of* f *associated with* $\mathbf{x}^*$

Definition 2.11 *Let* $\mathbf{x}^*$ *be a local minimizer of* $-f$. *The discrete basin of* $-f$ *at* $\mathbf{x}^*$ *is called a discrete hill of* f *at* $\mathbf{x}^*$.

Definition 2.12 *Let* $S_L = \{\mathbf{x} \in X : f(\mathbf{x}) < f(\mathbf{x}^*)\}$ *and* $S_U = \{\mathbf{x} \in X : f(\mathbf{x}) \geq f(\mathbf{x}^*)\}$.

2.2.2 Generic Discrete Filled Function Approach

The fundamental concept of a discrete filled function method can be explained as follows. An initial point is chosen and a local search is applied to find an initial discrete local minimizer. Then, an auxiliary function, called a filled function, is constructed at this local minimizer, where the local minimizer of the original function becomes a local maximizer of the filled function. By minimizing the filled function, either an improved discrete local minimizer is found or the boundary of the feasible region is reached. The discrete local minimizer of the filled function usually becomes a new starting point for minimizing the original objective with the hope of finding an improved point compared to the first local minimizer. A new filled function is constructed at this improved point. The process is repeated until no improved local minimizer of the earlier filled function can be found. The final discrete local minimizer is then taken as an approximation of the global minimizer.

If a local minimizer of the filled function cannot be found after repeated searches terminate on the boundary of the box constrained feasible region, the parameters defining the filled functions are adjusted and the search is repeated. This adjustment of the parameters continues until the parameters reach their predetermined bounds; the best solution obtained so far is taken as the global minimizer. Note that some filled functions have one parameter (such as those in [38, 127, 161]), while the rest are equipped with two parameters. The latter filled functions often have one parameter which is partially dependent on the other and this requires additional steps when tuning the parameters in order to satisfy the required convergence criteria. Note that each filled function discussed here has unique characteristics. The complexity of each filled function is also dependent on its associated algorithm, as discussed in detail in the following sections.

We present the generic framework of a discrete filled function algorithm. The main algorithm requires repeated searches for a local minimum. Thus, we state the local search as a separate algorithm (Algorithm 2.1 below). The global algorithm involves repeated construction of an auxiliary function in the hope of escaping basins associated with local minimizers.

Algorithm 2.1 *Discrete Steepest Descent Method*

1. *Choose an initial point* $\mathbf{x} \in X$.

2. *If* $\mathbf{x}$ *is a local minimizer of* f*, then stop. Otherwise, find the discrete steepest descent direction* $\mathbf{d}^* \in \mathcal{D}(\mathbf{x})$ *of* f.

3. *Set* $\mathbf{x} := \mathbf{x} + \mathbf{d}^*$. *Go to Step 2.*

Remark 2.1 *Note that some methods in the literature, namely those in [127, 165], merely require a discrete descent direction at Step 2, rather than a discrete steepest descent direction.*

Algorithm 2.2 *Discrete Filled Function Method*

1. *Initialization.*
   Set the bounds of each parameter in the formulation of the discrete filled function.
   Initialize the parameters.
   Choose suitable reduction or increment strategies for each parameter.
   Choose an initial starting point $\mathbf{x}_0 \in X$.

2. *Local search of the original function.*
   Starting from $\mathbf{x}_0$*, minimize* $f(\mathbf{x})$ *using Algorithm 2.1 to obtain a local minimizer* $\mathbf{x}^*$ *of* f.

3. *Neighbourhood search.*
   (a) Identify the neighbourhood of $\mathbf{x}^*$ *as* $N(\mathbf{x}^*) = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q\}$*, where* q *is the total number of points in* $N(\mathbf{x}^*)$*,* $q \leq 2n$*. Set* $\ell = 1$.
   (b) Define the current point, $\mathbf{x}_c := \mathbf{w}_\ell$.

4. *Local search of the discrete filled function.*
   Let $G_{\mathbf{x}^*}$ *denote the discrete filled function associated with* $\mathbf{x}^*$.
   Minimize $G_{\mathbf{x}^*}$ *using Algorithm 2.1 starting from* $\mathbf{x}_c$.
   Let $\acute{\mathbf{x}}$ *be the obtained local minimizer of* $G_{\mathbf{x}^*}$.

5. *Checking the status of* $\acute{\mathbf{x}}$.

   If $f(\acute{\mathbf{x}}) < f(\mathbf{x}^*)$, *set* $\mathbf{x}_0 := \acute{\mathbf{x}}$ *and go to Step 2. Otherwise, go to Step 6.*

6. *Checking other search directions.*

   At this point, the algorithms in [106, 107, 162] will adjust the parameters of the filled function and return to Step 4 if $\acute{\mathbf{x}} \in X \backslash \tilde{X}$.

   Otherwise, along with most of the remaining algorithms, they set $\ell := \ell + 1$.

   If $\ell \leq q$, *all of the algorithms then return directly to Step 3(b).*

   Otherwise, the parameters of the filled function are adjusted and ℓ *is reset to 1 before returning to Step 3(b).*

   If all the parameters of the filled function exceed their prescribed bounds anywhere in this step, the current value of $\mathbf{x}^*$ *is taken as the global minimizer.*

Remark 2.2 *Some methods in the literature, such as [127, 128, 162], replace* $N(\mathbf{x}^*)$ *in Step 3 with* $M = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q\}$, *where* $\mathbf{w}_i,\ i = 1, \ldots, q$, *are randomly chosen from* X. q *also needs to be chosen by the user in this case.*

Remark 2.3 *Some algorithms [38, 106, 107, 128, 160] do not require a local minimizer of* $G_{\mathbf{x}^*}$ *in Step 4. Instead, in the attempt to reduce* $G_{\mathbf{x}^*}$, *if any point* $\mathbf{x}_k$ *is found such that* $f(\mathbf{x}_k) < f(\mathbf{x}^*)$, *they set* $\mathbf{x}_0 := \mathbf{x}_k$ *and go back to Step 2.*

Remark 2.4 *Minimization of both* f *and* $G_{\mathbf{x}^*}$ *is carried out over* X, *except in [161], where* f *is minimized over* Λ *while* $G_{\mathbf{x}^*}$ *is minimized over* X. *Note that this variation is only relevant for nonlinearly constrained problems, though.*

Remark 2.5 *Note that the methods in [38, 106, 107, 162] define* X *via upper and lower bounds on the variables as well as a set of linear inequality constraints.*

Remark 2.6 *A slightly different approach is proposed in [106] for Step 4. If* f *and* $G_{\mathbf{x}^*}$ *share at least one common descent direction, the authors choose a steepest descent direction which results in the maximum reduction for* $f + G_{\mathbf{x}^*}$. *If such a direction does not exist, the method reverts to find a steepest descent direction for* $G_{\mathbf{x}^*}$ *only.*

For a clearer picture on how the filled function algorithm works, we consider an illustrative example in the next subsection.

2.2.3 Illustrative Example

$$\min\ f(\mathbf{x}) = 2x_1^2 - 1.05x_1^4 + \frac{1}{6}x_1^6 - x_1x_2 + x_2^2, \tag{2.6}$$

$$\text{s.t.}\quad x_i = \frac{y_i}{1000}, \quad -2000 \leq y_1 \leq 2000, \quad -1500 \leq y_2 \leq 1500, \quad y_1, y_2 \text{ integers}.$$

Problem (2.6) is a 3-hump back camel function in [19] which has 1.2007001×10^7 feasible points. This box constrained problem has a known global minimum solution at $\mathbf{x}^*_{\text{global}} = [0,0]^\top$ with $f(\mathbf{x}^*_{\text{global}}) = 0$. The discrete filled function method in [106] is used to solve this problem. The algorithm begins with a point $\mathbf{x}_0 = [1.500, 1.500]^\top$ with $f(\mathbf{x}_0) = 1.0828125$. By using the discrete steepest descent method, an initial local minimizer of $\mathbf{x}_1^* = [1.748, 0.874]^\top$ is found with $f(\mathbf{x}_1^*) = 0.2986396$. Next, a discrete filled function, $G_{\mathbf{x}_1^*}$, is constructed at $\mathbf{x}_1^*$. Starting with a point in $N(\mathbf{x}_1^*)$, $\mathbf{x}_c = [1.749, 0.874]^\top$, Algorithm 2.1 is used to minimize $G_{\mathbf{x}_1^*}$ and a local minimizer, $\acute{\mathbf{x}} = [0.302, 0.535]^\top$, with $f(\acute{\mathbf{x}}) = 0.2984554$, is found. Since $f(\acute{\mathbf{x}}) < f(\mathbf{x}_1^*)$, the original function f is minimized once more, starting at $\mathbf{x}_0 = \acute{\mathbf{x}}$, and the second local minimizer $\mathbf{x}_2^* = [0,0]^\top$, with $f(\mathbf{x}_2^*) = 0$, is obtained. Next, a new discrete filled function $G_{\mathbf{x}_2^*}$ is constructed at $\mathbf{x}_2^* = [0,0]^\top$. A neighbourhood point of $\mathbf{x}_2^* = [0,0]^\top$, namely $\mathbf{x}_c = [1,0]^\top$, is chosen, and $G_{\mathbf{x}_2^*}$ is minimized starting at $\mathbf{x}_c$. The local minimizer of $G_{\mathbf{x}_2^*}$ is a vertex, $\acute{\mathbf{x}} = [2.000, 1.500]^\top$, but $f(\acute{\mathbf{x}}) > f(\mathbf{x}_2^*)$. Other searches for a minimum of $G_{\mathbf{x}_2^*}$ in a lower basin are then carried out, starting from $[0,1]^\top$, $[-1,0]^\top$, and $[0,-1]^\top$, respectively. Since none of these yield an improved point, the parameter of $G_{\mathbf{x}_2^*}$ is adjusted. The revised $G_{\mathbf{x}_2^*}$ is then minimized once more starting from each of these neighbourhood points in turn. When no local minimizer of $G_{\mathbf{x}_2^*}$ in a lower basin is found and the termination criteria is met, $\mathbf{x}_2^* = [0,0]^\top$ is taken to be the global solution.

In the next section, we discuss and analyze various discrete filled function methods from the literature.

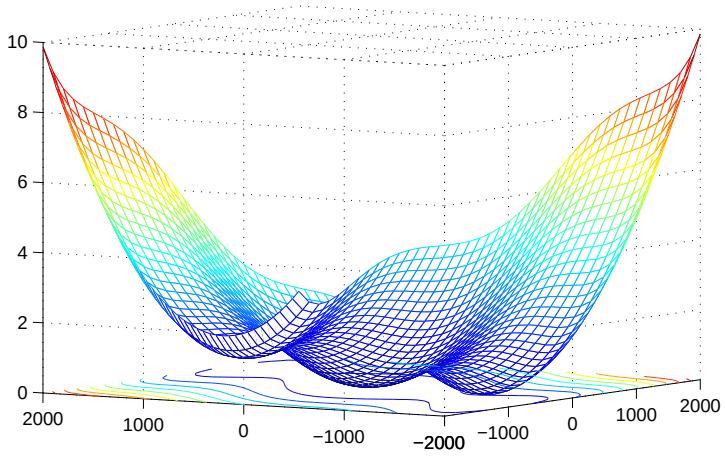

Figure 2.1: The 3-Hump Back Camel Function.

2.3 Discrete Filled Function Methods

2.3.1 Discrete Filled Function in Zhu [165]

Zhu is believed to be the first researcher to adapt the continuous filled function approach directly for solving discrete optimization problems. Let $\mathbf{x}^*$ denote the current discrete local minimizer. A filled function dependent on parameters θ and p is defined as

$$G_{\theta,p,\mathbf{x}^*}(\mathbf{x}) = \frac{1}{\theta + f(\mathbf{x})} \exp\left(\frac{\| \mathbf{x} - \mathbf{x}^* \|^2}{-p^2}\right). \tag{2.7}$$

Assuming that p and θ are chosen so that

$$0 < \theta + f(\mathbf{x}^*) < h$$

and

$$p^2 \ln\left(\frac{\theta + \bar{f}}{\theta + f(\mathbf{x}^*)}\right) < 1,$$

where $\bar{f}$ is an upper bound of f over X and $h \leq \min\{|f(\mathbf{x}_1) - f(\mathbf{x}_2)| : f(\mathbf{x}_1) \neq f(\mathbf{x}_2),\ \mathbf{x}_j \in X,\ j = 1, 2\}$, the filled function (2.7) has the following properties:

- $G_{\theta,p,\mathbf{x}^*}(\mathbf{x}^* + \mathbf{d}) < G_{\theta,p,\mathbf{x}^*}(\mathbf{x}^*)$, for all $\mathbf{d} \in \mathcal{D}(\mathbf{x}^*)$.
- Given $f(\mathbf{x}_1) \geq f(\mathbf{x}^*)$, $f(\mathbf{x}_2) \geq f(\mathbf{x}^*)$, and $\| \mathbf{x}_2 - \mathbf{x}^* \|^2 < \| \mathbf{x}_1 - \mathbf{x}^* \|^2$, $G_{\theta,p,\mathbf{x}^*}(\mathbf{x}_1) < G_{\theta,p,\mathbf{x}^*}(\mathbf{x}_2)$ (i.e. if f increases, $G_{\theta,p,\mathbf{x}^*}$ decreases).

- For any $\mathbf{x} \in X$,
  - $G_{\theta,p,\mathbf{x}^*}(\mathbf{x}) < 0 \iff f(\mathbf{x}) < f(\mathbf{x}^*)$;
  - $G_{\theta,p,\mathbf{x}^*}(\mathbf{x}) > 0 \iff f(\mathbf{x}) \geq f(\mathbf{x}^*)$.
- Let $\mathbf{x}_1 \in X$ be such that $f(\mathbf{x}_1) \geq f(\mathbf{x}^*)$.
  - If there exists $\mathbf{d} \in \mathcal{D}(\mathbf{x}_1)$ such that $G_{\theta,p,\mathbf{x}^*}(\mathbf{x}_1 + \mathbf{d}) < 0$; or
  - If $|\{\mathbf{d} \in \mathcal{D}(\mathbf{x}_1) : \mathbf{x}_1 + \mathbf{d} \in X\}| = n$ and there exists $\mathbf{d} \in \mathcal{D}(\mathbf{x}_1)$ such that $G_{\theta,p,\mathbf{x}^*}(\mathbf{x}_1 + \mathbf{d}) < G_{\theta,p,\mathbf{x}^*}(\mathbf{x}_1)$; or
  - If $|\{\mathbf{d} \in \mathcal{D} : \mathbf{x}_1 + \mathbf{d} \in X\}| > n$;

  then there exists some $\mathbf{d} \in \mathcal{D}(\mathbf{x}_1)$ such that $G_{\theta,p,\mathbf{x}^*}(\mathbf{x}_1+\mathbf{d}) < G_{\theta,p,\mathbf{x}^*}(\mathbf{x}_1) < G_{\theta,p,\mathbf{x}^*}(\mathbf{x}^*)$.
- Any discrete local minimizer of the discrete filled function $G_{\theta,p,\mathbf{x}^*}$ must be in the set S_L or $\tilde{X}$.

Zhu suggests that the algorithm should stop when all searches for a minimum of $G_{\theta,p,\mathbf{x}^*}$ starting in $N(\mathbf{x}^*)$ terminate at vertices without finding an improved point of f. Note that the algorithm in [165] does not require updating of the parameters θ and p. Thus, the final $\mathbf{x}^*$ is assumed to be the global minimum. Two numerical examples are demonstrated to test the efficiency of this filled function. However, the disadvantage of his method is that it is almost impossible to find a negative filled function value that would indicate that a point in a lower basin exists. This is because the discrete filled function contains an exponential term, making it ill conditioned and also leading to poor efficiency as noted in [106]. In addition, it is difficult to determine suitable values of h and $\bar{f}$, thus making it difficult to find suitable values for parameters θ and p.

2.3.2 Discrete Filled Function in Ng, Zhang, Li & Tian [107]

A new discrete filled function with improved theoretical properties was proposed in [107] several years later. Recall that B^* denotes a discrete basin of f that contains

the current discrete local minimizer $\mathbf{x}^*$. According to [107], a function $G_{\mu,\rho,\mathbf{x}^*}$ is defined to be a discrete filled function of f at $\mathbf{x}^*$ if it satisfies the following:

- $\mathbf{x}^*$ is a strict local maximizer of $G_{\mu,\rho,\mathbf{x}^*}$;
- $G_{\mu,\rho,\mathbf{x}^*}$ has no discrete local minimizers in B^* or in any discrete basin of f higher than B^*;
- If f has a discrete basin B^{**} at $\mathbf{x}^{**}$ which is lower than B^*, then there is a discrete point $\acute{\mathbf{x}} \in B^{**}$ that minimizes $G_{\mu,\rho,\mathbf{x}^*}$ on a connected discrete path $\{\mathbf{x}^*, \ldots, \acute{\mathbf{x}}, \ldots, \mathbf{x}^{**}\}$ in X.

The discrete filled function proposed in [107] is

$$G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}) = f(\mathbf{x}^*) - \min[f(\mathbf{x}^*), f(\mathbf{x})] - \rho \parallel \mathbf{x} - \mathbf{x}^* \parallel^2 + \mu\{\max[0, f(\mathbf{x}) - f(\mathbf{x}^*)]\}^2, \tag{2.8}$$

where ρ and μ are parameters which satisfy certain properties as detailed below.

- Recall that the meaning of $\mathcal{K}$ and $\mathcal{L}$ from Assumptions 2.1 and 2.2. Suppose that $\bar{\mathbf{x}} \in S_U$.
  - If $\rho > 0$ and $0 \leq \mu < \frac{\rho}{\mathcal{L}^2}$, then $G_{\mu,\rho,\mathbf{x}^*}(\bar{\mathbf{x}}) < 0 = G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}^*)$.
  - If $\rho > 0$ and $0 \leq \mu \leq \frac{\rho}{2\mathcal{K}^2\mathcal{L}^2}$, then for each $\bar{\mathbf{d}} \in \mathcal{D}(\bar{\mathbf{x}})$ such that $f(\bar{\mathbf{x}} + \bar{\mathbf{d}}) \geq f(\mathbf{x}^*)$ and $\parallel \bar{\mathbf{x}} + \bar{\mathbf{d}} - \mathbf{x}^* \parallel > \parallel \bar{\mathbf{x}} - \mathbf{x}^* \parallel$, $G_{\mu,\rho,\mathbf{x}^*}(\bar{\mathbf{x}} + \bar{\mathbf{d}}) < G_{\mu,\rho,\mathbf{x}^*}(\bar{\mathbf{x}}) < 0 = G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}^*)$.
- If $\rho > 0$ and $0 \leq \mu < \frac{\rho}{\mathcal{L}^2}$, then $\mathbf{x}^*$ is a strict local maximizer of $G_{\mu,\rho,\mathbf{x}^*}$. If $\mathbf{x}^*$ is a global minimizer of f, then $G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}^*) < 0$, for all $\mathbf{x} \in X \backslash \mathbf{x}^*$.
- Let $\mathbf{x}_1,\ \mathbf{x}_2,\ \mathbf{x}^*$ be three distinct points in X. If $\parallel \mathbf{x}_2 - \mathbf{x}^* \parallel > \parallel \mathbf{x}_1 - \mathbf{x}^* \parallel$, then $1 \leq \dfrac{\parallel \mathbf{x}_2 - \mathbf{x}_1 \parallel}{\parallel \mathbf{x}_2 - \mathbf{x}^* \parallel - \parallel \mathbf{x}_1 - \mathbf{x}^* \parallel} < 2\mathcal{K}^2$.
- Let $\mathbf{x}_1,\ \mathbf{x}_2 \in X$ be two points such that $0 < \parallel \mathbf{x}_1 - \mathbf{x}^* \parallel < \parallel \mathbf{x}_2 - \mathbf{x}^* \parallel$ and $f(\mathbf{x}^*) \leq f(\mathbf{x}_1) \leq f(\mathbf{x}_2)$. If $\rho > 0$ and $0 \leq \mu \leq \frac{\rho}{2\mathcal{K}^2\mathcal{L}^2}$, then
  - $G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_2) < G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_1) < 0 = G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}^*)$;

- $G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}^*)$ has no local minimizers in B^* or in any discrete basin of f higher than B^*.

- For every $\acute{\mathbf{x}}, \mathbf{x}^* \in X$, there exists $\mathbf{d} \in E$ such that $\| \acute{\mathbf{x}} + \mathbf{d} - \mathbf{x}^* \| > \| \acute{\mathbf{x}} - \mathbf{x}^* \|$.

- Let $\mathbf{x}^* \in X$ and $\acute{\mathbf{x}} \in X$ be the local minimizers of f and $G_{\mu,\rho,\mathbf{x}^*}$, respectively. If $\rho > 0$ and $0 \leq \mu \leq \frac{\rho}{2\mathcal{K}^2\mathcal{L}^2}$, then

  - $f(\acute{\mathbf{x}} + \acute{\mathbf{d}}) < f(\mathbf{x}^*)$ for all $\acute{\mathbf{d}} \in \mathcal{D}(\acute{\mathbf{x}})$ when $f(\acute{\mathbf{x}}) \geq f(\mathbf{x}^*)$.
  - $\acute{\mathbf{x}}$ is in a basin B^{**} (associated with a local minimum $\mathbf{x}^{**}$) of f which is lower than basin B^* (associated with $\mathbf{x}^*$) .

Both μ and ρ are initialized as 1. This filled function ensures that a local minimizer of $G_{\mu,\rho,\mathbf{x}^*}$ is either a better point in a lower basin or a vertex of X. It is not necessary to find the minimizer of $G_{\mu,\rho,\mathbf{x}^*}$ if a point $\mathbf{x}_k$ with $f(\mathbf{x}_k) < f(\mathbf{x}^*)$ is found in Step 4 of Algorithm 2.2. Since $\mathbf{x}_k$ is an improved point, the algorithm sets $\mathbf{x}_0 := \mathbf{x}_k$ and returns to Step 2 to minimize the original function f. If the minimizer of $G_{\mu,\rho,\mathbf{x}^*}$ is not a vertex, μ is reduced via $\mu := \mu/10$ and $G_{\mu,\rho,\mathbf{x}^*}$ is minimized once more starting at the same $\mathbf{x}_c$. When no improved point is found after the minimization process for $G_{\mu,\rho,\mathbf{x}^*}$ ends up at a vertex, then ℓ is increased by 1 and $G_{\mu,\rho,\mathbf{x}^*}$ is minimized once more starting with the updated $\mathbf{x}_c$. If $\ell > q$, ρ is reduced. The algorithm terminates when the lower bound of ρ, ρ_L, is met. Several test problems were investigated in [107] and the proposed discrete filled function method was shown to be efficient in solving problems involving up to 200 variables. Note that ρ_L was set to 1 for the computations in [107] and further reduction of ρ was not necessary since all test problems yielded the global solution when $\rho = 1$. According to one of the characteristics of this filled function, $\acute{\mathbf{x}}$, the local minimizer of $G_{\mu,\rho,\mathbf{x}^*}$, lies on a discrete path $\{\mathbf{x}^*, \ldots, \acute{\mathbf{x}}, \ldots, \mathbf{x}^{**}\}$ in X that connects the current basin B^* at $\mathbf{x}^*$ to a lower basin B^{**}. However, the properties of this filled function do not guarantee that $\acute{\mathbf{x}}$ is a true minimizer of the original

function. A revised discrete filled function is proposed in [106] to overcome this difficulty.

2.3.3 Discrete Filled Function in Ng, Li & Zhang [106]

Based on the work in [107], a new discrete filled function $G_{\mu,\rho,\mathbf{x}^*}$ at $\mathbf{x}^*$ is defined as follows:

$$G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}) = A_\mu(f(\mathbf{x}) - f(\mathbf{x}^*)) - \rho \parallel \mathbf{x} - \mathbf{x}^* \parallel, \tag{2.9}$$
$$A_\mu(y) = y \cdot \mu \left[(1-c) \left(\frac{1-c\mu}{\mu - c\mu} \right)^{-y/\omega} + c \right],$$

where $\omega > 0$ is a sufficiently small number and $0 < c \leq 1$ is a constant. The function $G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x})$ is a discrete filled function when certain conditions of the parameters μ and ρ are satisfied as detailed in the following conditions:

- $\mathbf{x}^*$ is a strict local maximizer of $G_{\mu,\rho,\mathbf{x}^*}$.
- $G_{\mu,\rho,\mathbf{x}^*}$ has no local minimizer in the set $S_U \backslash \tilde{X}$.
- $\mathbf{x}^{**} \in X \backslash \tilde{X}$ is a local minimizer of f if and only if $\mathbf{x}^{**}$ is a local minimizer of $G_{\mu,\rho,\mathbf{x}^*}$. In short, $\mathbf{x}^{**} \in S_L$.
- If $\rho > 0$ and $0 < \mu < \min\{1, \frac{\rho}{\mathcal{L}}\}$, then $\mathbf{x}^*$ is a strict local maximizer of $G_{\mu,\rho,\mathbf{x}^*}$. If $\mathbf{x}^*$ is a global minimizer of f, then $G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}) < 0$ for all $\mathbf{x} \in X \setminus \mathbf{x}^*$.
- Let $\bar{\mathbf{d}} \in \mathcal{D}(\bar{\mathbf{x}})$ be a feasible direction at $\bar{\mathbf{x}} \in S_U$ such that $\parallel \bar{\mathbf{x}} + \bar{\mathbf{d}} - \mathbf{x}^* \parallel > \parallel \bar{\mathbf{x}} - \mathbf{x}^* \parallel$. If $\rho > 0$ and $0 < \mu < \min\{1, \frac{\rho}{2\mathcal{K}^2\mathcal{L}}\}$, then $G_{\mu,\rho,\mathbf{x}^*}(\bar{\mathbf{x}} + \bar{\mathbf{d}}) < G_{\mu,\rho,\mathbf{x}^*}(\bar{\mathbf{x}}) < 0 = G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}^*)$.
- Let $\mathbf{x}^{**}$ be a strict local minimizer of f with $f(\mathbf{x}^{**}) < f(\mathbf{x}^*)$. If $\rho > 0$ is sufficiently small and $0 < \mu < 1$, then $\mathbf{x}^{**}$ is a strict local minimizer of $G_{\mu,\rho,\mathbf{x}^*}$.
- Let $\acute{\mathbf{x}}$ be a strict local minimizer of $G_{\mu,\rho,\mathbf{x}^*}$ and $\bar{\mathbf{d}} \in \mathcal{D}(\acute{\mathbf{x}})$ be a feasible direction at $\acute{\mathbf{x}}$ such that $\parallel \acute{\mathbf{x}} + \bar{\mathbf{d}} - \mathbf{x}^* \parallel > \parallel \acute{\mathbf{x}} - \mathbf{x}^* \parallel$. If $\rho > 0$ is sufficiently small and $0 < \mu < \min\{1, \frac{\rho}{2\mathcal{K}^2\mathcal{L}}\}$, then $\acute{\mathbf{x}}$ is a local minimizer of f.

- Assume that every local minimizer of f is strict. Suppose that $\rho > 0$ is sufficiently small and $0 < \mu < \min\{1, \frac{\rho}{2\mathcal{K}^2\mathcal{L}}\}$. Then, $\mathbf{x}^{**} \in X \setminus \tilde{X}$ is a local minimizer of f with $f(\mathbf{x}^{**}) < f(\mathbf{x}^*)$ if and only if $\mathbf{x}^{**}$ is a local minimizer of $G_{\mu,\rho,\mathbf{x}^*}$.

This is an improved version of the discrete filled function in [107], to ensure $\acute{\mathbf{x}}$ coincides with $\mathbf{x}^{**}$. In other words, every local minimizer of the discrete filled function $G_{\mu,\rho,\mathbf{x}^*}$ is also a local minimizer for the original function f. Both μ and ρ are initialized as 0.1. The parameter μ is reduced if $\acute{\mathbf{x}}$ is not an improved point and by setting $\mu := \mu/10$ and returning to Step 3(a). If $\acute{\mathbf{x}}$ is not an improved point and a vertex of X, set $\ell := \ell + 1$ and return to Step 3(b), unless $\ell > q$ in which case ρ is adjusted. Similar to [107], the algorithm for minimizing $G_{\mu,\rho,\mathbf{x}^*}$ exits prematurely when an improved point $\mathbf{x}_k$ with $f(\mathbf{x}_k) < f(\mathbf{x}^*)$ is found in Step 4 of Algorithm 2.2. The algorithm sets $\mathbf{x}_0 := \mathbf{x}_k$ and returns to Step 2 to minimize the original function f in this case. Note that a direction which yields the greatest improvement of $f + G_{\mu,\rho,\mathbf{x}^*}$ is chosen when minimizing $G_{\mu,\rho,\mathbf{x}^*}$, assuming that a direction for improving f and $G_{\mu,\rho,\mathbf{x}^*}$ simultaneously does exist. If such a direction does not exist, the algorithm chooses the steepest descent direction such that $G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d}^*) < G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c)$. The algorithm terminates when $\rho_L = 0.1$. Note that ρ is fixed at 0.1, since all test problems in [106] readily yield a global solution when $\rho = 0.1$. The filled function in (2.9) is shown to increase computational efficiency when compared with that in [107]. Several test problems with up to 1.38×10^{104} feasible points were solved using this method.

2.3.4 Discrete Filled Function in Yang & Liang [160]

A two parameter exponential filled function,

$$G_{a,b,\mathbf{x}^*}(\mathbf{x}) = \frac{1}{a + \parallel \mathbf{x} - \mathbf{x}^* \parallel} \Upsilon\Big(\max\{f(\mathbf{x}) - f(\mathbf{x}^*) + b, 0\} \Big), \tag{2.10}$$

where

$$\Upsilon(y) = \begin{cases} \exp(-a/y), & \text{if } y \neq 0, \\ 0, & \text{if } y = 0. \end{cases}$$

is introduced in [160]. Let S_M represent the set of discrete local minimizers of f, $a < 0$, and

$$0 < b < \max_{\substack{\mathbf{x}^*, \mathbf{x}^{**} \in S_M \\ f(\mathbf{x}^{**}) < f(\mathbf{x}^*)}} \Big(f(\mathbf{x}^*) - f(\mathbf{x}^{**}) \Big).$$

$G_{a,b,\mathbf{x}^*}$ is a discrete filled function of f if $G_{a,b,\mathbf{x}^*}(\mathbf{x})$ has the following properties:

- $\mathbf{x}^*$ is a strict discrete local maximizer of $G_{a,b,\mathbf{x}^*}$.
- $G_{a,b,\mathbf{x}^*}$ has no discrete local minimizers in S_U.
- If $\mathbf{x}^*$ is not a discrete global minimizer of f, then $G_{a,b,\mathbf{x}^*}$ does have a discrete minimizer $\acute{\mathbf{x}} \in S_L$.
- For any $\mathbf{x}, \mathbf{x}^* \in X$, there exists $\mathbf{d} \in \mathcal{D}(\mathbf{x})$ such that $\| \mathbf{x} + \mathbf{d} - \mathbf{x}^* \| < \| \mathbf{x} - \mathbf{x}^* \|$.
- Let $\mathbf{x}_1,\ \mathbf{x}_2,\ \mathbf{x}^*$ be three distinct points in X. If $\| \mathbf{x}_2 - \mathbf{x}^* \| > \| \mathbf{x}_1 - \mathbf{x}^* \|$, then $\dfrac{\| \mathbf{x}_1 - \mathbf{x}^* \|}{\| \mathbf{x}_2 - \mathbf{x}^* \|} < 1 - \dfrac{1}{2\mathcal{K}^2}$.
- For any $\mathbf{x}_1,\ \mathbf{x}_2 \in X$, if
  - $\| \mathbf{x}_2 - \mathbf{x}^* \| > \| \mathbf{x}_1 - \mathbf{x}^* \|$,
  - $f(\mathbf{x}_1) \geq f(\mathbf{x}^*)$, and
  - $f(\mathbf{x}_2) - f(\mathbf{x}^*) + b > 0$,

  then $G_{a,b,\mathbf{x}^*}(\mathbf{x}_2) < G_{a,b,\mathbf{x}^*}(\mathbf{x}_1)$.

The parameters a and b are initialized as 0.01 and 1, respectively. When all the search directions from $\mathbf{x}^*$ have been utilized but no improved point of f is found (i.e. $\ell > q$), the user either sets $b := b/10$ and $a := a/10$ or $a := a/10$ only as long as $a > 10^{-7}$. The algorithm terminates when $b \leq 10^{-5}$. Note that it is not necessary to find the minimizer of $G_{a,b,\mathbf{x}^*}$ for this algorithm. As long as a point $\mathbf{x}_k$ with $f(\mathbf{x}_k) < f(\mathbf{x}^*)$ is found when minimizing $G_{a,b,\mathbf{x}^*}$, the algorithm reverts to minimizing the original function f. As in [107], a local minimizer of this filled function is not guaranteed to be a true local minimizer of the original function f.

2.3.5 Discrete Filled Function in Shang & Zhang [127]

A third exponential filled function is suggested in [127]. Let $\mathbf{x}^*$ be the current local minimizer and choose any $\mathbf{x}_0$ such that $f(\mathbf{x}_0) \geq f(\mathbf{x}^*)$. According to [127], $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$ is called a discrete filled function of f at $\mathbf{x}^*$ if $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$ has the following properties:

- $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$ has no local minimizer in $S_U \backslash \{\mathbf{x}_0\}$ and $\mathbf{x}_0$ is not necessarily a local minimizer of $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$;
- If $\mathbf{x}^*$ is not a global minimizer of f, there exists a local minimizer $\acute{\mathbf{x}} \in S_L$ of $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$ such that $f(\acute{\mathbf{x}}) < f(\mathbf{x}^*)$.

A discrete filled function, with parameter ϖ, is defined as follows:

$$G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}(\mathbf{x}) = \zeta(\| \mathbf{x}-\mathbf{x}_0 \|)-\xi(\varpi(1-\exp(-[\min\{f(\mathbf{x})-f(\mathbf{x}^*),0\}]^2))), \quad (2.11)$$

where $\varpi > 0$ and the prefixed point $\mathbf{x}_0$ satisfies $f(\mathbf{x}_0) \geq f(\mathbf{x}^*)$. In addition, the functions $\zeta(t)$ and $\xi(t)$ have the following characteristics:

- $\zeta(t)$ and $\xi(t)$ are strictly increasing for any $t \in [0, +\infty)$;
- $\zeta(0) = 0$ and $\xi(0) = 0$;
- $\xi(t) \to C > 0$ as $\mathbf{x} \to +\infty$, where $C \geq \max_{\mathbf{x}\in X} \zeta(\| \mathbf{x} - \mathbf{x}_0 \|)$.

In addition, the following conditions hold for $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$:

- For any $\mathbf{x} \in X$, if $\mathbf{x} \neq \mathbf{x}_0$, there exists $\mathbf{d} \in \mathcal{D}(\mathbf{x})$ such that $\| \mathbf{x} + \mathbf{d} - \mathbf{x}_0 \| < \| \mathbf{x} - \mathbf{x}_0 \|$.
- $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$ has no local minimizer in $S_U \backslash \{\mathbf{x}_0\}$ for any $\varpi > 0$.
- Suppose $S_L \neq \emptyset$. If ϖ satisfies $\varpi > \dfrac{\xi^{-1}(C)\exp([f(\bar{\mathbf{x}}^*) - f(\mathbf{x}^*)]^2)}{\exp([f(\bar{\mathbf{x}}^*) - f(\mathbf{x}^*)]^2) - 1}$, where $\bar{\mathbf{x}}^*$ is a global minimizer of f, then $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$ has a local minimizer in S_L.

- Suppose that ε is a small positive constant and ϖ satisfies $\varpi > \dfrac{\xi^{-1}(C)\exp(\varepsilon^2)}{\exp(\varepsilon^2)-1}$. Then, given any $\mathbf{x}^*$ of f such that $f(\mathbf{x}^*) \geq f(\bar{\mathbf{x}}^*) + \varepsilon$, where $\bar{\mathbf{x}}^*$ is a global minimizer of f, $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$ has at least one local minimizer in S_L.

Instead of performing a neighbourhood search in Step 3 of Algorithm 2.2, the implementation in [127] uses any initial point on the boundary of X to minimize $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$. In [127], the parameter ϖ is fixed to $400.5(10\sqrt{n}+1)$, where n is the dimension of a problem. For each subsequent initial point drawn from the boundary of X, $i := i+1$ and the algorithm terminates when $i = 10^n$. Every local minimizer of $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$ is assumed to be an improved point (Step 5 of Algorithm 2.2 is by-passed). Though this filled function has only one fixed parameter, the local search of $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$ can become computationally intensive due to the large number of initial points that may need to be tested before the termination criteria is met. A nonlinear box constrained problem with up to 1.71×10^5 feasible points was solved in [127]. Similar to the methods in [107, 160], a local minimizer of the filled function $G_{\varpi,\mathbf{x}_0,\mathbf{x}^*}$ is not necessarily a local minimizer of the original function f. Furthermore, a prefixed point $\mathbf{x}_0$ is required at the beginning of the algorithm, resulting in the minimization process typically converging to $\mathbf{x}_0$ rather than an improved point of the original function. A refined formulation of this filled function is suggested in [128].

2.3.6 Discrete Filled Function in Shang & Zhang [128]

Let

$$G_{\delta,q,\mathbf{x}^*}(\mathbf{x}) = \frac{\ln(1+q\max(f(\mathbf{x})-f(\mathbf{x}^*)+\delta,0))}{1+\parallel \mathbf{x}-\mathbf{x}^* \parallel} \tag{2.12}$$

be a discrete filled function of f with $q > 0$,

$$0 < \delta < \min_{\substack{\mathbf{x}_1,\mathbf{x}_2 \in X \\ \mathbf{x}_1 \neq \mathbf{x}_2}} |f(\mathbf{x}_1) - f(\mathbf{x}_2)|.$$

It has the following properties:

- $\mathbf{x}^*$ is a strict local maximizer of $G_{\delta,q,\mathbf{x}^*}$.

- If $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ and $\mathbf{x} \neq \mathbf{x}^*$, then $\mathbf{x}$ is not a local minimizer of $G_{\delta,q,\mathbf{x}^*}$.
- If $\mathbf{x}^*$ is not a global minimizer of $f(\mathbf{x})$, there exists a local minimizer $\acute{\mathbf{x}}$ of $G_{\delta,q,\mathbf{x}^*}$ in S_L.
- If $\mathbf{x}_1, \mathbf{x}_2 \in X$ are two distinct points which satisfy the following conditions:
  - $f(\mathbf{x}_1) \geq f(\mathbf{x}^*)$ and $f(\mathbf{x}_2) \geq f(\mathbf{x}^*)$, and
  - $\| \mathbf{x}_1 - \mathbf{x}^* \| > \| \mathbf{x}_2 - \mathbf{x}^* \| > 0$,

  then $G_{\delta,q,\mathbf{x}^*}(\mathbf{x}_1) < G_{\delta,q,\mathbf{x}^*}(\mathbf{x}_2)$.
- If $\mathbf{x}_1, \mathbf{x}_2 \in X$ are two distinct points which satisfy the following conditions:
  - $f(\mathbf{x}_2) \geq f(\mathbf{x}^*) > f(\mathbf{x}_1)$, and
  - $\| \mathbf{x}_1 - \mathbf{x}^* \| > \| \mathbf{x}_2 - \mathbf{x}^* \| > 0$,

  then, $G_{\delta,q,\mathbf{x}^*}(\mathbf{x}_1) < G_{\delta,q,\mathbf{x}^*}(\mathbf{x}_2)$.

This filled function overcomes the prefixed point issue in [127] to ensure a better point of the original function is attained and suggests an additional parameter. The initial settings for δ and q are 1 and 100, respectively. A random initial point in X is used to minimize $G_{\delta,q,\mathbf{x}^*}$ instead of a neighbourhood point as suggested in Step 3 of Algorithm 2.2. If no local minimizer of $G_{\delta,q,\mathbf{x}^*}$ is found along the search from this random point, another initial point in X is drawn and $i := i + 1$. When $i > 2n$, the algorithm sets $q := 10q$ as long as $q < 10^5$. Otherwise, the algorithm sets $\delta := \delta/10$ and $q := q_0$ in Step 6 of Algorithm 2.2. Then, i is reset to 1 and $G_{\delta,q,\mathbf{x}^*}$ is minimized again from the same starting point with the new parameter values. Similar to [160], it is not necessary to find a minimizer of $G_{\delta,q,\mathbf{x}^*}$. The algorithm terminates when $\delta < 10^{-5}$ and $\ell = 2n$, where n refers to the dimension of the problem. Two test problems, with up to 1.1739×10^{52} feasible points were solved in [128]. Since a local minimizer of this filled function is not necessarily a

local minimizer of the original function f, further computation is needed to find the local minimizer of f in a lower basin for each local minimizer of $G_{\delta,q,\mathbf{x}^*}(\mathbf{x})$ found.

2.3.7 Discrete Filled Function in Yang & Zhang [162]

Suppose $\varphi(t)$ is a continuously differentiable function satisfying the following conditions:

- $\varphi(t) = \vartheta$ when $t \geq \epsilon$; $\varphi(t) = -\vartheta$ when $t \leq -\epsilon$.
- $\acute{\varphi}(t) \geq 0, \ -\epsilon \leq t \leq \epsilon$.
- $\varphi(0) = 0$.

Suppose also that a function $\eta(t)$ satisfies $\eta(0) = 0$ and $\acute{\eta}(t) > 0$, for $t \geq 0$. The filled function in [162] is given by

$$G_{\epsilon,\nu,\mathbf{x}^*}(\mathbf{x}) = \eta(\| \mathbf{x} - \mathbf{x}_0 \|)\varphi(f(\mathbf{x}) - f(\mathbf{x}^*) + \nu), \tag{2.13}$$

where $\mathbf{x}_0$ is an arbitrary point in X, ϑ is a positive constant, and both ϵ and ν are problem-dependent parameters. The properties for this discrete filled function are as follows:

- The function $G_{\epsilon,\nu,\mathbf{x}^*}$ has no discrete local minimizer except at $\mathbf{x}_0$ in the region $S_1 = \{\mathbf{x} \in X : f(\mathbf{x}) \geq f(\mathbf{x}^*) + \epsilon - \nu\}$, where $\epsilon \geq \nu$.
- If $\nu = 0$, $G_{\epsilon,\nu,\mathbf{x}^*}(\mathbf{x})$ has no discrete local minimizer except at $\mathbf{x}_0$ in $S_2 = \{\mathbf{x} \in X : f(\mathbf{x}) \geq f(\mathbf{x}^*) + \epsilon\}$.
- If $\nu = \epsilon$, $G_{\epsilon,\nu,\mathbf{x}^*}(\mathbf{x})$ has no discrete local minimizer except at $\mathbf{x}_0$ in S_U.
- Given $\nu = 0$ or $\nu = \epsilon$, if ϵ is sufficiently small and $\mathbf{x}^*$ is not a discrete global minimizer of f, then $G_{\epsilon,\nu,\mathbf{x}^*}(\mathbf{x})$ does have a discrete local minimizer $\acute{\mathbf{x}}$ in S_L.
- If $\mathbf{x}^*$ is a global minimizer of f, then $\mathbf{x}_0$ is the unique discrete global minimizer of $G_{\epsilon,\nu,\mathbf{x}^*}(\mathbf{x})$ with $\nu > 0$.

The functions $\eta(t)$ and $\varphi(t)$ in (2.13) must be chosen carefully to ensure computational reliability and efficiency. As a guide, polynomial functions are suggested in [162] for both $\eta(t)$ and $\varphi(t)$. Based on the characteristics of this filled function, ϵ and ν are initialized as 1.0 and 0, respectively, so that there exists a local minimizer of $G_{\epsilon,\nu,\mathbf{x}^*}$ in a lower basin. The disadvantage of this filled function is that it depends heavily on the initial point $\mathbf{x}_0$ in computing $G_{\epsilon,\nu,\mathbf{x}^*}$. Thus, $\mathbf{x}_0$ has to be chosen carefully and plays a crucial role in finding a local minimizer of $G_{\epsilon,\nu,\mathbf{x}^*}$ such that $f(\bar{\mathbf{x}}^*) \leq f(\mathbf{x}_1^*) + \epsilon$, where $\bar{\mathbf{x}}^*$ is the global minimum of the original function. If a local minimizer of the filled function in a lower basin cannot be determined, then $\mathbf{x}_0$ is taken as its local minimizer, with suitable values of ϵ and ν, or $\mathbf{x}_0$ is assumed to be the global solution of the original function, which is not likely to happen in practice. The algorithm terminates when $\epsilon < 0.0001$. Several test problems with up to 200 variables have been solved using this filled function method as reported in [162].

2.3.8 Discrete Filled Function in Gu & Wu [38]

Gu and Wu propose the discrete filled function

$$G_{\varrho,\mathbf{x}^*}(\mathbf{x}) = \frac{1}{\| \mathbf{x} - \mathbf{x}^* \|^2 + 1} E_\varrho\big(f(\mathbf{x}) - f(\mathbf{x}^*)\big) + F_\varrho\big(f(\mathbf{x}) - f(\mathbf{x}^*)\big), \qquad (2.14)$$

where

$$E_\varrho(y) = \begin{cases} 0, & y \leq -\varrho, \\ -\frac{2y^3}{\varrho^3} - \frac{3y^2}{\varrho^2} + 1, & -\varrho < y \leq 0, \\ 1, & y > 0, \end{cases}$$

and

$$F_\varrho(y) = \begin{cases} y + \varrho, & y \leq -\varrho, \\ \frac{(\varrho-2)y^3}{\varrho^3} + \frac{(\varrho-3)y^2}{\varrho^2} + 1, & -\varrho < y \leq 0, \\ 1, & y > 0. \end{cases}$$

Define $\beta_0 = \min_{\mathbf{x} \in S_L}(f(\mathbf{x}^*) - f(\mathbf{x}))$. If the function parameter ϱ satisfies

$$0 < \varrho \leq \beta_0,$$

then the following results hold.

- For all $\mathbf{x} \in X$, $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ is equivalent to $G_{\varrho,\mathbf{x}^*}(\mathbf{x}) > 1$.
- $\mathbf{x}^*$ is not a global minimizer of f if and only if $S_L \neq \emptyset$ and $\beta_0 > 0$.
- $\mathbf{x} \in S_L$ is equivalent to $G_{\varrho,\mathbf{x}^*}(\mathbf{x}) \leq 0$.
- $\mathbf{x}^*$ is a strict discrete local maximizer of $G_{\varrho,\mathbf{x}^*}$.
- If $\mathbf{x}^*$ is not a global minimizer of f, then there exists a discrete local minimizer of $G_{\varrho,\mathbf{x}^*}$, denoted by $\acute{\mathbf{x}}$.
- $\acute{\mathbf{x}}$ is either in S_L or $\tilde{X}$.
- Given $\mathbf{x}_1, \mathbf{x}_2 \in S_U$, $G_{\varrho,\mathbf{x}^*}(\mathbf{x}_1) > G_{\varrho,\mathbf{x}^*}(\mathbf{x}_2)$ is equivalent to $\| \mathbf{x}_1 - \mathbf{x}^* \| < \| \mathbf{x}_2 - \mathbf{x}^* \|$.

The parameter ϱ is initialized as 1. It is updated in Step 6 of Algorithm 2.2 by setting $\varrho := \varrho/10$ when all available search directions at $\mathbf{x}^*$ have been used (i.e. $\ell > q$) but no improved point of f is found. The algorithm terminates when $\varrho = 10^{-5}$. The one-parameter filled function suggested here guarantees that the minimizer of $G_{\varrho,\mathbf{x}^*}$ is also a minimizer of f. Based on this approach, a refined algorithm which is capable of dealing directly with nonlinear constraints is proposed in [161].

2.3.9 Discrete Filled Function in Yang, Wu & Bai [161]

An extended study of the filled function method in [38] is given in [161] to deal with the nonlinear constrained problem (2.3). A one-parameter discrete filled function is defined as

$$G_{r,\mathbf{x}^*}(\mathbf{x}) = \left(\frac{1}{\| \mathbf{x} - \mathbf{x}^* \|^2 + 1} + 1\right)\Gamma\left(H_r(f(\mathbf{x}) - f(\mathbf{x}^*)) + \sum_{i=1}^{m} H_r(g_i(\mathbf{x}) - r)\right), \tag{2.15}$$

where

$$H_r(y) = \begin{cases} 0, & y \leq -r, \\ \frac{(r-2)y^3}{r^3} + \frac{(2r-3)y^2}{r^2} + y + 1, & -r < y \leq 0, \\ y + 1, & y > 0, \end{cases}$$

and

$$\Gamma(y) = \begin{cases} 0, & y \leq 0.5, \\ -16y^3 + 36y^2 - 24y + 5, & 0.5 < y \leq 1, \\ 1, & y > 1, \end{cases}$$

Let $\check{\beta} = \min\{\beta_0, \beta_1\}$, where

$$\beta_0 = \min_{x \in S_L} \big(f(\mathbf{x}^*) - f(\mathbf{x})\big)$$

and

$$\beta_1 = \min_{x \in X \backslash \Lambda} \max_{i \in \{1,\ldots,m\}} g_i(\mathbf{x}).$$

If the parameter r satisfies

$$0 < r \leq \check{\beta},$$

$G_{r,\mathbf{x}^*}$ is said to be a discrete filled function at $\mathbf{x}^*$ and the following properties hold.

- $\mathbf{x}^*$ is a strict discrete local maximizer of $G_{r,\mathbf{x}^*}$ on X.
- If $\mathbf{x}^*$ is not a global minimizer of f, then there exists a $\acute{\mathbf{x}} \in S_L$ such that $\acute{\mathbf{x}}$ is a discrete local minimizer of $G_{r,\mathbf{x}^*}$.
- Any discrete local minimizer of $G_{r,\mathbf{x}^*}$ is either in S_L or in $\tilde{X}$.
- Given $\mathbf{x}_1, \mathbf{x}_2 \in X \backslash S_L$, $G_{r,\mathbf{x}^*}(\mathbf{x}_1) > G_{r,\mathbf{x}^*}(\mathbf{x}_2)$ if and only if $\| \mathbf{x}_1 - \mathbf{x}^* \| < \| \mathbf{x}_2 - \mathbf{x}^* \|$.
- $\mathbf{x} \in X \backslash S_L$ if and only if $G_{r,\mathbf{x}^*}(\mathbf{x}) > 1$.
- $\mathbf{x} \in S_L$ if and only if $G_{r,\mathbf{x}^*}(\mathbf{x}) = 0$.

Unlike the other filled functions discussed earlier, this filled function is capable of solving constrained nonlinear problems directly. Sets Λ and X are the feasible regions of f and $G_{r,\mathbf{x}^*}$, respectively. Note that the algorithm as stated in [161] is incomplete without justifying how to handle the non-feasibility issue of $\mathbf{x}_0$ if $\mathbf{x}_0 \in X \backslash \Lambda$ happens to be used at the beginning of the algorithm. Based on correspondence with the main author in [161], we suggest an additional preliminary step before Step 1 in Algorithm 2.2 to check if $\mathbf{x}_0 \in \Lambda$ before minimizing f. If

this condition is satisfied, then continue with Step 1 in Algorithm 2.2. Otherwise, set $\mathbf{x}^* := \mathbf{x}_0$ and jump directly to Step 3 in Algorithm 2.2. The nature of the filled function is such that a minimum point of it must lie in Λ. Thus, a $\mathbf{x}_0 \in \Lambda$ can be readily obtained.

Since the local minimizer of the discrete filled function has to be tested for feasibility with respect to the original function, it is not guaranteed to be a local minimizer of f. Thus, further computation is needed for this single-parameter filled function approach for each minimizer of the filled function found. The parameter r is set as 1 at the beginning of the algorithm, reduced by $r := r/10$ when $\ell > q$ in Step 6 of Algorithm 2.2, and the algorithm terminates when $r = 10^{-5}$.

2.4 Solutions of Test Problems

In this section, we select several promising discrete filled function methods from those described in the previous section, based on their theoretical properties and algorithms. These functions are tested on several benchmark problems: Colville's function [44], Goldstein and Price's function [37], Beale's singular function [102], Powell's singular function [102], and Rosenbrock's function [120]. Note that our aim is to simply compare the efficiency of different discrete filled function methods without necessarily solving high dimensional problems. Note, though, that these methods have been demonstrated to solve problems involving up to 200 variables [106, 107]. These algorithms are as follows:

- Algorithm A extracted from [107];
- Algorithm B extracted from [106];
- Algorithm C extracted from [160];
- Algorithm D extracted from [161].

The performance of each of the filled function methods used in solving the test problem is summarized in the following subsections. Note that we set $\rho_L = 0.001$

in Algorithms A and B, to be more confident of obtaining a global solution when solving these test problems, rather than $\rho_L = 1$ and 0.1 as suggested in [107] and [106], respectively. Note further that we construct a look-up table to store each objective function value computed so far to avoid repeated calculation of the objective function for the same point. This modification was introduced in view of the proposed application of discrete filled function methods to mixed discrete optimization problems in later chapters, where each function evaluation is computationally expensive. The final optimal solution found for each algorithm is recorded by $\mathbf{x}^*_{\text{final}}$ with its corresponding objective value $f(\mathbf{x}^*_{\text{final}})$. The total number of original function evaluations, the total number of discrete filled function evaluations, and the ratio of the average number of original function evaluations to reach the global solution to the total number of feasible points are represented in Table 2.1-2.5 by E_f, E_G, and R_E, respectively.

2.4.1 Problem 1: Colville's Function

$$\begin{aligned} \min\ f(\mathbf{x}) \ &= \ 100\big(x_2 - x_1^2\big)^2 + \big(1 - x_1\big)^2 + 90\big(x_4 - x_3^2\big)^2 + \big(1 - x_3\big)^2 \\ &\quad + 10.1\Big[\big(x_2 - 1\big)^2 + \big(x_4 - 1\big)^2\Big] + 19.8\big(x_2 - 1\big)\big(x_4 - 1\big), \end{aligned}$$

$$\text{s.t.} \quad -10 \le x_i \le 10, \quad x_i \text{ integer}, \quad i = 1, 2, 3, 4.$$

This box constrained problem has 1.94481×10^5 feasible points. The global minimum solution is $\mathbf{x}^*_{\text{global}} = [1, 1, 1, 1]^\top$ with $f(\mathbf{x}^*_{\text{global}}) = 0$. Six starting points were considered for the algorithms, namely $[1, 1, 0, 0]^\top$, $[1, 1, 1, 1]^\top$, $[-10, 10, -10, 10]^\top$, $[-10, -5, 0, 5]^\top$, $[-10, 0, 0, -10]^\top$, and $[0, 0, 0, 0]^\top$. All discrete filled function algorithms succeeded in finding the global minimum from all starting points. A summary of the computational results is displayed in Table 2.1. Numerical results show that Algorithm B has the smallest total number of original function evaluations, and the average R_E is 0.008635805.

Table 2.1: Numerical Results of Problem 1.

Algorithm	$\mathbf{x}_0$	$\mathbf{x}^*_{\text{final}}$	$f(\mathbf{x}^*_{\text{final}})$	E_f	E_G	R_E
A	$[1,1,0,0]^\top$	$[1,1,1,1]^\top$	0	2095	7058	0.010772261
	$[1,1,1,1]^\top$	$[1,1,1,1]^\top$	0	2086	7037	0.010725984
	$[-10,10,-10,10]^\top$	$[1,1,1,1]^\top$	0	3940	10603	0.020259048
	$[-10,-5,0,5]^\top$	$[1,1,1,1]^\top$	0	2192	7056	0.011271024
	$[-10,0,0,-10]^\top$	$[1,1,1,1]^\top$	0	2226	7059	0.011445848
	$[0,0,0,0]^\top$	$[1,1,1,1]^\top$	0	2102	7060	0.010808254
B	$[1,1,0,0]^\top$	$[1,1,1,1]^\top$	0	1426	5097	0.007332336
	$[1,1,1,1]^\top$	$[1,1,1,1]^\top$	0	1422	5076	0.007311768
	$[-10,10,-10,10]^\top$	$[1,1,1,1]^\top$	0	2674	5979	0.013749415
	$[-10,-5,0,5]^\top$	$[1,1,1,1]^\top$	0	1567	5134	0.008057342
	$[-10,0,0,-10]^\top$	$[1,1,1,1]^\top$	0	1557	5098	0.008005923
	$[0,0,0,0]^\top$	$[1,1,1,1]^\top$	0	1431	5099	0.007358045
C	$[1,1,0,0]^\top$	$[1,1,1,1]^\top$	0	3041	35243	0.015636489
	$[1,1,1,1]^\top$	$[1,1,1,1]^\top$	0	2867	34570	0.014741800
	$[-10,10,-10,10]^\top$	$[1,1,1,1]^\top$	0	4608	39849	0.023693831
	$[-10,-5,0,5]^\top$	$[1,1,1,1]^\top$	0	3842	37147	0.019755143
	$[-10,0,0,-10]^\top$	$[1,1,1,1]^\top$	0	3174	35253	0.016320360
	$[0,0,0,0]^\top$	$[1,1,1,1]^\top$	0	3051	35254	0.015687908
D	$[1,1,0,0]^\top$	$[1,1,1,1]^\top$	0	1615	15973	0.008304153
	$[1,1,1,1]^\top$	$[1,1,1,1]^\top$	0	1435	15312	0.007378613
	$[-10,10,-10,10]^\top$	$[1,1,1,1]^\top$	0	4145	21660	0.021313136
	$[-10,-5,0,5]^\top$	$[1,1,1,1]^\top$	0	2569	17483	0.013209517
	$[-10,0,0,-10]^\top$	$[1,1,1,1]^\top$	0	1748	15992	0.008988025
	$[0,0,0,0]^\top$	$[1,1,1,1]^\top$	0	1625	15993	0.008355572

2.4.2 Problem 2: Goldstein and Price's Function

$$\min \ f(\mathbf{x}) = g(\mathbf{x})h(\mathbf{x})$$

$$\text{s.t.} \quad x_i = \frac{y_i}{1000} \quad -2000 \leq y_i \leq 2000, \quad y_i \text{ integer}, \quad i = 1, 2,$$

where

$$g(\mathbf{x}) = 1 + \big(x_1 + x_2 + 1\big)^2\big(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\big),$$

and

$$h(\mathbf{x}) = 30 + \big(2x_1 - 3x_2\big)^2\big(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\big).$$

This box constrained problem has 1.6008001×10^7 feasible points. The global minimum solution is $\mathbf{x}^*_{\text{global}} = [0, -1]^\top$ with $f(\mathbf{x}^*_{\text{global}}) = 3$. Six starting points were considered in the computational tests, these being $[2, -2]^\top$, $[0, -1]^\top$, $[-2, -2]^\top$, $[-0.5, -1]^\top$, $[1, -1.5]^\top$, and $[1, -1]^\top$. A summary of the computational results is given in Table 2.2. All algorithms succeeded in finding the global minimum from all starting points, where Algorithm B is shown to be the most efficient method. This method succeeded in identifying the global minimum solution with an average of 22249 function evaluations. The average R_E is 0.0013899.

2.4.3 Problem 3: Beale's Function

$$\begin{aligned}\min \ f(\mathbf{x}) \ = \ & \Big[1.5 - x_1\big(1 - x_2\big)\Big]^2 + \Big[2.25 - x_1\big(1 - x_2^2\big)\Big]^2 \\ & + \Big[2.625 - x_1\big(1 - x_2^3\big)\Big]^2,\end{aligned}$$

$$\text{s.t.} \quad x_i = \frac{y_i}{1000} \quad -10000 \leq y_i \leq 10000, \quad y_i \text{ integer}, \quad i = 1, 2.$$

Table 2.2: Numerical Results of Problem 2.

Algorithm	$\mathbf{x}_0$	$\mathbf{x}^*_{\text{final}}$	$f(\mathbf{x}^*_{\text{final}})$	E_f	E_G	R_E
A	$[2,-2]^\top$	$[0,-1]^\top$	3	51234	217255	0.003200525
	$[0,-1]^\top$	$[0,-1]^\top$	3	47189	217255	0.002947838
	$[-2,-2]^\top$	$[0,-1]^\top$	3	53675	217255	0.003353011
	$[-0.5,-1]^\top$	$[0,-1]^\top$	3	47189	217255	0.002947838
	$[1,-1.5]^\top$	$[0,-1]^\top$	3	50723	217255	0.003168603
	$[1,-1]^\top$	$[0,-1]^\top$	3	47189	217255	0.002947838
B	$[2,-2]^\top$	$[0,-1]^\top$	3	25041	151356	0.001564280
	$[0,-1]^\top$	$[0,-1]^\top$	3	18995	151356	0.001186594
	$[-2,-2]^\top$	$[0,-1]^\top$	3	24472	151356	0.001528736
	$[-0.5,-1]^\top$	$[0,-1]^\top$	3	20475	151356	0.001279048
	$[1,-1.5]^\top$	$[0,-1]^\top$	3	22533	151356	0.001407609
	$[1,-1]^\top$	$[0,-1]^\top$	3	21978	151356	0.001372938
C	$[2,-2]^\top$	$[0,-1]^\top$	3	50028	1170105	0.003125187
	$[0,-1]^\top$	$[0,-1]^\top$	3	45983	1170105	0.002872501
	$[-2,-2]^\top$	$[0,-1]^\top$	3	52469	1170105	0.003277673
	$[-0.5,-1]^\top$	$[0,-1]^\top$	3	45983	1170105	0.002872501
	$[1,-1.5]^\top$	$[0,-1]^\top$	3	49517	1170105	0.003093266
	$[1,-1]^\top$	$[0,-1]^\top$	3	45983	1170105	0.002872501
D	$[2,-2]^\top$	$[0,-1]^\top$	3	48030	623910	0.003000375
	$[0,-1]^\top$	$[0,-1]^\top$	3	43985	623910	0.002747688
	$[-2,-2]^\top$	$[0,-1]^\top$	3	50475	623910	0.003153111
	$[-0.5,-1]^\top$	$[0,-1]^\top$	3	43985	623910	0.002747688
	$[1,-1.5]^\top$	$[0,-1]^\top$	3	47519	623910	0.002968453
	$[1,-1]^\top$	$[0,-1]^\top$	3	43985	623910	0.002747688

This box constrained problem has 4.00040001×10^8 feasible points. The global minimum solution is $\mathbf{x}^*_{\text{global}} = [3, 0.5]^\top$ with $f(\mathbf{x}^*_{\text{global}}) = 0$. Six starting points were considered in the tests: $[10, -10]^\top$, $[9.997, -6.867]^\top$, $[0, -1]^\top$, $[1, 1]^\top$, $[-2, 2]^\top$, and $[0, 0]^\top$. A summary of the computational results is shown in Table 2.3. Only Algorithms A and B consistently succeeded in identifying the global minimum with the average number of function evaluations being 119722.2 and 358077.3, respectively. Note that Algorithm B is more efficient than Algorithm A, where the average R_E is 0.000299275, compared to 0.000895104. As for Algorithms C and D, both yielded local minimizers close to the global solution: $[3.015, 0.504]^\top$, $[2.989, 0.497]^\top$, $[3.004, 0.501]^\top$, and $[2.996, 0.499]^\top$. A possible reason for this failure to converge tot he global solution may be that our implementation calls on neighbourhood points in Step 3 in a different order to that in other implementations.

2.4.4 Problem 4: Powell's Singular Function

$$\begin{aligned} \min\ f(\mathbf{x}) \ &= \ \big(x_1 + 10x_2\big)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 \\ &\quad + 10(x_1 - x_4)^4, \end{aligned}$$

$$\text{s.t.} \quad x_i = \frac{y_i}{1000} \quad -10000 \le y_i \le 10000, \quad y_i \text{ integer}, \quad i = 1, 2, 3, 4.$$

This box constrained problem has 1.60032×10^{17} feasible points. The global minimum is at $\mathbf{x}^*_{\text{global}} = [0, 0, 0, 0]^\top$ with $f(\mathbf{x}^*_{\text{global}}) = 0$. Six starting points were used in the tests: $[10, 10, 10, 10]^\top$, $[-10, -10, -10, -10]^\top$, $[10, -10, -10, 10]^\top$, $[1, -1, -1, 1]^\top$, $[-10, 1, 0, 5]^\top$, and $[0, 0, 0, 0]^\top$. All methods succeeded in identifying the global minimum. Table 2.4 summaries the computational results. Numerical experiments suggest that Algorithm B has the smallest total number of original function evaluations, and the average R_E is 7.01735×10^{-15}.

Table 2.3: Numerical Results of Problem 3.

Algorithm	$\mathbf{x}_0$	$\mathbf{x}^*_{\text{final}}$	$f(\mathbf{x}^*_{\text{final}})$	E_f	E_G	R_E
A	$[10,-10]^\top$	$[3,0.5]^\top$	0	408190	1781788	0.001020373
	$[9.997,-6.867]^\top$	$[3,0.5]^\top$	0	410442	1781788	0.001026002
	$[0,-1]^\top$	$[3,0.5]^\top$	0	415309	1781788	0.001038169
	$[1,1]^\top$	$[3,0.5]^\top$	0	216860	1140046	0.000542096
	$[-2,2]^\top$	$[3,0.5]^\top$	0	219484	1140046	0.000548655
	$[0,0]^\top$	$[3,0.5]^\top$	0	478179	2049532	0.001195328
B	$[10,-10]^\top$	$[3,0.5]^\top$	0	119997	1310251	0.000299963
	$[9.997,-6.867]^\top$	$[3,0.5]^\top$	0	121489	1310251	0.000303692
	$[0,-1]^\top$	$[3,0.5]^\top$	0	129333	1310251	0.000323300
	$[1,1]^\top$	$[3,0.5]^\top$	0	107219	723603	0.000268021
	$[-2,2]^\top$	$[3,0.5]^\top$	0	105842	723603	0.000264579
	$[0,0]^\top$	$[3,0.5]^\top$	0	134453	776637	0.000336099
C	$[10,-10]^\top$	$[3.015,0.504]^\top$	0.0000376	100002	128430	0.000249980
	$[9.997,-6.867]^\top$	$[3.015,0.504]^\top$	0.0000376	100002	123335	0.000249980
	$[0,-1]^\top$	$[3.015,0.504]^\top$	0.0000376	100001	111165	0.000249978
	$[1,1]^\top$	$[2.989,0.497]^\top$	0.0000211	100001	199532	0.000249978
	$[-2,2]^\top$	$[2.989,0.497]^\top$	0.0000211	100001	202671	0.000249978
	$[0,0]^\top$	$[2.989,0.497]^\top$	0.0000211	100002	206268	0.000249980
D	$[10,-10]^\top$	$[3.004,0.501]^\top$	0.00000255	386183	2610857	0.000965361
	$[9.997,-6.867]^\top$	$[3.004,0.501]^\top$	0.00000255	388440	2610857	0.000971003
	$[0,-1]^\top$	$[3.004,0.501]^\top$	0.00000255	393307	2610857	0.000983169
	$[1,1]^\top$	$[2.996,0.499]^\top$	0.00000257	257134	2110006	0.000642771
	$[-2,2]^\top$	$[2.996,0.499]^\top$	0.00000257	276458	2110006	0.000691076
	$[0,0]^\top$	$[3.004,0.501]^\top$	0.00000255	494215	2711826	0.001235414

Table 2.4: Numerical Results of Problem 4.

Algorithm	$\mathbf{x}_0$	$\mathbf{x}^*_{\text{final}}$	$f(\mathbf{x}^*_{\text{final}})$	E_f	E_G	R_E
A	$[10,10,10,10]^\top$	$[0,0,0,0]^\top$	0	1874	7248	1.17102×10^{-14}
	$[-10,-10,-10,-10]^\top$	$[0,0,0,0]^\top$	0	1928	7247	1.20476×10^{-14}
	$[10,-10,-10,10]^\top$	$[0,0,0,0]^\top$	0	1825	7248	1.14040×10^{-14}
	$[1,-1,-1,1]^\top$	$[0,0,0,0]^\top$	0	1742	7248	1.08853×10^{-14}
	$[-10,1,0,5]^\top$	$[0,0,0,0]^\top$	0	1807	7247	1.12915×10^{-14}
	$[0,0,0,0]^\top$	$[0,0,0,0]^\top$	0	1732	7243	1.08228×10^{-14}
B	$[10,10,10,10]^\top$	$[0,0,0,0]^\top$	0	1160	5350	7.24855×10^{-15}
	$[-10,-10,-10,-10]^\top$	$[0,0,0,0]^\top$	0	1179	5349	7.36728×10^{-15}
	$[10,-10,-10,10]^\top$	$[0,0,0,0]^\top$	0	1131	5350	7.06734×10^{-15}
	$[1,-1,-1,1]^\top$	$[0,0,0,0]^\top$	0	1067	5350	6.66742×10^{-15}
	$[-10,1,0,5]^\top$	$[0,0,0,0]^\top$	0	1140	5349	7.12358×10^{-15}
	$[0,0,0,0]^\top$	$[0,0,0,0]^\top$	0	1061	5345	6.62992×10^{-15}
C	$[10,10,10,10]^\top$	$[0,0,0,0]^\top$	0	2777	36061	1.73528×10^{-14}
	$[-10,-10,-10,-10]^\top$	$[0,0,0,0]^\top$	0	2536	34605	1.58468×10^{-14}
	$[10,-10,-10,10]^\top$	$[0,0,0,0]^\top$	0	2759	36061	1.72403×10^{-14}
	$[1,-1,-1,1]^\top$	$[0,0,0,0]^\top$	0	2612	36061	1.63217×10^{-14}
	$[-10,1,0,5]^\top$	$[0,0,0,0]^\top$	0	2420	34605	1.51220×10^{-14}
	$[0,0,0,0]^\top$	$[0,0,0,0]^\top$	0	2342	34594	1.46346×10^{-14}
D	$[10,10,10,10]^\top$	$[0,0,0,0]^\top$	0	2043	17777	1.27662×10^{-14}
	$[-10,-10,-10,-10]^\top$	$[0,0,0,0]^\top$	0	1744	16478	1.08978×10^{-14}
	$[10,-10,-10,10]^\top$	$[0,0,0,0]^\top$	0	2048	17777	1.27974×10^{-14}
	$[1,-1,-1,1]^\top$	$[0,0,0,0]^\top$	0	1874	17777	1.17102×10^{-14}
	$[-10,1,0,5]^\top$	$[0,0,0,0]^\top$	0	1620	16478	1.01230×10^{-14}
	$[0,0,0,0]^\top$	$[0,0,0,0]^\top$	0	1542	16458	9.63557×10^{-15}

2.4.5 Problem 5: Rosenbrock's Function

$$\min\ f(\mathbf{x}) = \sum_{i=1}^{n-1} \Big[100\big(x_{i+1} - x_i^2\big)^2 + \big(1 - x_i\big)^2\Big],$$

$$\text{s.t.} \quad -5 \leq x_i \leq 5, \quad x_i \text{ integer}, \quad i = 1, 2, \ldots, n.$$

This box constrained problem has 1.08347×10^{26} feasible points for $n = 25$. The global minimum is at $\mathbf{x}^*_{\text{global}} = [1, \ldots, 1]^\top$ with $f(\mathbf{x}^*_{\text{global}}) = 0$. Six starting points were considered in the simulations: $[0, \ldots, 0]^\top$, $[3, \ldots, 3]^\top$, $[-5, \ldots, -5]^\top$, $[2, -2, \ldots, 2, -2, 2]^\top$, $[3, -3, \ldots, 3, -3, 3]^\top$, and $[5, -5, \ldots, 5, -5, 5]^\top$. All algorithms succeeded in identifying the global minimum for most of the starting points used. A summary of the computational results is displayed in Table 2.5. Clearly, Algorithm B has the least total number of original function evaluations and the average R_E is 1.87477×10^{-21}.

2.4.6 Comparison with Literature Results

Table 2.6 shows the average values of a number of original function evaluations for an algorithm to terminate and compares this with the results from the literature. Since these test problems were not solved in [161], we compare our numerical results with those in [107], [106], and [160] only. Recall that in our implementations of these algorithms, we construct a look-up table to store each objective function value computed so far to avoid repeated calculation of the objective function. Consequently, our implementations show a significantly lower number of function evaluations when compared to the results found in the literature. We note that in our implementation of the various algorithms, searches for a local minimum of the filled function may be initialized with different starting points than those used in the implementations published previously. This is because either the order in which the neighbourhood of $\mathbf{x}^*$ is to be tested is not specified or the starting points are not confined to the neighbourhood $N(\mathbf{x}^*)$ and are chosen randomly within the feasible region. This difference may well influence the actual performance of an algorithm.

Table 2.5: Numerical Results of Problem 5.

Algorithm	$\mathbf{x}_0$	$\mathbf{x}^*_{\text{final}}$	$f(\mathbf{x}^*_{\text{final}})$	E_f	E_G	R_E
A	$[0,\ldots,0]^\top$	$[1,\ldots,1]^\top$	0	211831	682050	1.95512×10^{-21}
	$[3,\ldots,3]^\top$	$[1,\ldots,1]^\top$	0	418536	898526	3.86292×10^{-21}
	$[-5,\ldots,-5]^\top$	$[1,\ldots,1]^\top$	0	217435	682050	2.00684×10^{-21}
	$[2,-2,\ldots,2,-2,2]^\top$	$[1,\ldots,1]^\top$	0	214231	682050	1.97727×10^{-21}
	$[3,-3,\ldots,3,-3,3]^\top$	$[1,\ldots,1]^\top$	0	510907	1006018	4.71547×10^{-21}
	$[5,-5,\ldots,5,-5,5]^\top$	$[1,\ldots,1]^\top$	0	512802	1006018	4.73296×10^{-21}
B	$[0,\ldots,0]^\top$	$[1,\ldots,1]^\top$	0	171072	444101	1.57893×10^{-21}
	$[3,\ldots,3]^\top$	$[1,\ldots,1]^\top$	0	312888	644091	2.88783×10^{-21}
	$[-5,\ldots,-5]^\top$	$[1,\ldots,1]^\top$	0	176624	444101	1.63017×10^{-21}
	$[2,-2,\ldots,2,-2,2]^\top$	$[1,\ldots,1]^\top$	0	173472	444101	1.60108×10^{-21}
	$[3,-3,\ldots,3,-3,3]^\top$	$[1,\ldots,1]^\top$	0	191402	563646	1.76656×10^{-21}
	$[5,-5,\ldots,5,-5,5]^\top$	$[1,\ldots,1]^\top$	0	193297	563646	1.78405×10^{-21}
C	$[0,\ldots,0]^\top$	$[0,\ldots,0]^\top$*	24	532603	3031547	4.91571×10^{-21}
	$[3,\ldots,3]^\top$	$[1,\ldots,1]^\top$	0	627360	2824273	5.79277×10^{-21}
	$[-5,\ldots,-5]^\top$	$[0,\ldots,0]^\top$*	24	538156	3031547	4.96696×10^{-21}
	$[2,-2,\ldots,2,-2,2]^\top$	$[0,\ldots,0]^\top$*	24	534952	3031547	4.93739×10^{-21}
	$[3,-3,\ldots,3,-3,3]^\top$	$[1,\ldots,1]^\top$	0	678295	2920682	6.26039×10^{-21}
	$[5,-5,\ldots,5,-5,5]^\top$	$[1,\ldots,1]^\top$	0	680190	2920682	6.27788×10^{-21}
D	$[0,\ldots,0]^\top$	$[0,\ldots,0]^\top$*	24	182636	1493376	1.68566×10^{-21}
	$[3,\ldots,3]^\top$	$[1,\ldots,1]^\top$	0	289538	1401000	2.67232×10^{-21}
	$[-5,\ldots,-5]^\top$	$[0,\ldots,0]^\top$*	24	188189	1493376	1.73691×10^{-21}
	$[2,-2,\ldots,2,-2,2]^\top$	$[0,\ldots,0]^\top$*	24	184985	1493376	1.70734×10^{-21}
	$[3,-3,\ldots,3,-3,3]^\top$	$[1,\ldots,1]^\top$	0	339380	1493460	3.13234×10^{-21}
	$[5,-5,\ldots,5,-5,5]^\top$	$[1,\ldots,1]^\top$	0	341275	1493460	3.14983×10^{-21}

*Remarks: The final solution is a local solution.

Table 2.6: A Comparison of Function Evaluations.

Problem	Algorithm	Our implementations	Results in [107]	Results in [106]	Results in [160]
1	A	2440.17	4263.11		
	B	1679.5		3767.78	
	C	3430.5			85705
	D	2189.5			
2	A	49533.17	111125.86		
	B	22249		68196.29	
	C	48327.17			2125511
	D	46329.83			
3	A	366914.3	939209.57		
	B	119368.8		444887.71	
	C	100001.5*			4861560
	D	365956.2*			
4	A	1818	7337207.5		
	B	1123		6731232	
	C	2574.333			155868850
	D	1811.8333			
5	A	347623.7	320610.44		
	B	203125.8		305712.11	
	C	598637.7			6282030
	D	254333.8			

*Remarks: The final solution is a local solution.

2.5 Concluding Remarks

Discrete filled function methods have shown promising results in finding globally optimal solutions in several benchmark problems as demonstrated in the previous section, thus confirming the applicability, reliability, and efficiency of this relatively recent global optimization technique. As can be seen from Table 2.6, Algorithm B is the most efficient method, yielding the lowest number of function evaluations for solving all test problems. Our intention is to adapt the technique to complex mixed discrete optimization problems where individual objective function evaluations are computationally expensive. Methods requiring the least number of function evaluations are important in solving such problems. In the next chapter, we propose some variations to Algorithm B to enhance the computational efficiency, before adapting it to solve discrete-valued optimal control problems in subsequent chapters.

Chapter 3

Variations of Discrete Filled Function Methods

This chapter summarizes some of our own ideas of how an existing discrete filled function algorithm may be modified to improve its performance. This is done with a view of finding the most suitable algorithm for our proposed technique of solving discrete-valued optimal control problems in the coming chapters. We adopt Algorithm B extracted from [106] in the previous chapter and propose five major variations to this algorithm. Each algorithm is tested on Colville's function and Rosenbrock's function as defined in Subsections 2.4.1 and 2.4.5, respectively. The performances of the proposed variations of the basic algorithm are summarized at the end of this chapter.

Before discussing the details of the proposed variations, we recall the following basic box constrained discrete optimization problem:

$$\min f(\mathbf{x}), \quad \text{s.t. } \mathbf{x} \in X,$$

where $X = \{\mathbf{x} \in \mathbb{Z}^n | \mathbf{x}_{i,\min} \leq \mathbf{x}_i \leq \mathbf{x}_{i,\max}\}$, $\mathbb{Z}^n$ is the set of integer points in $\mathbb{R}^n$, and $\mathbf{x}_{i,\min},\ \mathbf{x}_{i,\max},\ i = 1, \ldots, n$, are given bounds. Also, we recall the discrete steepest descent method from Subsection 2.2.2 as follows.

Algorithm 3.1 *Discrete Steepest Descent Method*

1. *Choose an initial point* $\mathbf{x} \in X$.

2. *If* $\mathbf{x}$ *is a local minimizer of* f*, then stop. Otherwise, find the discrete steepest descent direction* $\mathbf{d}^* \in \mathcal{D}(\mathbf{x})$ *of* f.

3. *Set* $\mathbf{x} := \mathbf{x} + \mathbf{d}^*$. *Go to Step 2.*

3.1 The Standard Algorithm

The following is the original filled function algorithm extracted from [106].

Algorithm 3.2 *Standard Algorithm*

1. *Initialize* $\mathbf{x}_0 \in X$, ρ_0, μ_0, $\rho_L > 0$, $0 < \hat{\rho} < 1$, *and* $0 < \hat{\mu} < 1$.
   Let $\rho := \rho_0$ *and* $\mu := \mu_0$.
   Choose an initial point $\mathbf{x}_0 \in X$.

2. *Starting from* $\mathbf{x}_0$, *minimize* $f(\mathbf{x})$ *using Algorithm 3.1 to obtain a local minimizer* $\mathbf{x}^*$ *of* f.

3. *(a) List the neighbouring points of* $\mathbf{x}^*$ *as* $N(\mathbf{x}^*) = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q\}$. *Set* $\ell := 1$.
   (b) Set the current switching point, $\mathbf{x}_c := \mathbf{w}_\ell$.

4. *(a) If there exists a direction* $\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)$ *such that* $f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}^*)$, *then set* $\mathbf{x}_0 := \mathbf{x}_c + \mathbf{d}$ *and go to Step 2. Otherwise, go to (b) below.*
   (b) Let $\mathcal{D}_1 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : \ f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}_c)$ *and* $G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$.
   If $\mathcal{D}_1 \neq \emptyset$, *set* $\mathbf{d}^* := \arg\min_{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)} \{f(\mathbf{x}_c + \mathbf{d}) + G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$.
   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to (c) below.*
   (c) Let $\mathcal{D}_2 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : \ G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$.

If $\mathcal{D}_2 \neq \emptyset$, set $\mathbf{d}^* := \arg\min_{\mathbf{d}\in\mathcal{D}(\mathbf{x}_c)}\{G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$.

Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ and go to (a) above. Otherwise, go to Step 5.

5. *Let $\acute{\mathbf{x}} = \mathbf{x}_c$ be the local minimizer of $G_{\mu,\rho,\mathbf{x}^*}$ obtained from Step 4.*

   (a) If $\acute{\mathbf{x}} \in \tilde{X}$, set $\ell := \ell + 1$. If $\ell > q$, go to Step 6. Otherwise, go to Step 3(b).

   (b) If $\acute{\mathbf{x}} \notin \tilde{X}$, reduce μ by setting $\mu := \hat{\mu}\mu$ and go to Step 4(b).

6. *Reduce ρ by setting $\rho := \hat{\rho}\rho$. If $\rho < \rho_L$, terminate the algorithm. The current $\mathbf{x}^*$ is taken as a global minimizer of the problem. Otherwise, set $\ell := 1$ and go to Step 3(b).*

Table 3.1 describes the numerical results from implementing Algorithm 3.2 for minimizing Rosenbrock's function with $n = 5$. The global minimum is $\mathbf{x}^*_{\text{global}} = [1, 1, 1, 1, 1]^\top$ with $f(\mathbf{x}^*_{\text{global}}) = 0$. Recall from Section 2.4 that the total number of original function evaluations, the total number of discrete filled function evaluations, and the ratio of the average number of original function evaluations to reach the global solution to the total number of feasible points are denoted in the table by E_f, E_G, and R_E, respectively. Note that we set $\rho_L = 0.001$ in the numerical computation to be more confident of obtaining a global solution when minimizing both test problems. Eleven starting points are considered in solving the problem. These are $[-5, -5, -5, -5, -5]^\top$, $[-4, -4, -4, -4, -4]^\top$, $[-3, -3, -3, -3, -3]^\top$, $[-2, -2, -2, -2, -2]^\top$, $[-1, -1, -1, -1, -1]^\top$, $[0, 0, 0, 0, 0]^\top$, $[1, 1, 1, 1, 1]^\top$, $[2, 2, 2, 2, 2]^\top$, $[3, 3, 3, 3, 3]^\top$, $[4, 4, 4, 4, 4]^\top$, and $[5, 5, 5, 5, 5]^\top$. The algorithm was able to determine the global solution from all starting points.

Recall that Colville's function has a minimum global $\mathbf{x}^*_{\text{global}} = [1, 1, 1, 1]^\top$ with $f(\mathbf{x}^*_{\text{global}}) = 0$. Six starting points are considered in Algorithm 3.2, namely $[1, 1, 0, 0]^\top$, $[1, 1, 1, 1]^\top$, $[-10, 10, -10, 10]^\top$, $[-10, -5, 0, 5]^\top$, $[-10, 0, 0, -10]^\top$, and $[0, 0, 0, 0]^\top$. The algorithm succeeded in finding the global minimum from all starting points, as displayed in Table 2.1.

Table 3.1: Results of Algorithm 3.2 - Rosenbrock's Function.

$\mathbf{x}_0$	E_f	E_G	R_E
$[-5,-5,-5,-5,-5]^\top$	1436	4475	0.45952
$[-4,-4,-4,-4,-4]^\top$	1435	4475	0.45920
$[-3,-3,-3,-3,-3]^\top$	1395	4475	0.44640
$[-2,-2,-2,-2,-2]^\top$	1354	4475	0.43328
$[-1,-1,-1,-1,-1]^\top$	1314	4475	0.42048
$[0,0,0,0,0]^\top$	1274	4475	0.40768
$[1,1,1,1,1]^\top$	1252	4415	0.40064
$[2,2,2,2,2]^\top$	1271	4415	0.40672
$[3,3,3,3,3]^\top$	1646	5720	0.52672
$[4,4,4,4,4]^\top$	1666	5720	0.53312
$[5,5,5,5,5]^\top$	1664	5720	0.53248

3.2 The First Variation

We replace the set $N(\mathbf{x}^*)$ in Step 3 of the previous algorithm with a set which is just outside of the immediate neighbourhood of $\mathbf{x}^*$. Then, an additional step is introduced after Step 3 to test whether an improved point exists amongst the points in this alternative set. If so, the first improved point identified is used as the starting point to minimize f. The motivation behind this algorithm is to seek an improved point more efficiently than Algorithm 3.2 by bypassing those points which are in the immediate neighbourhood of $\mathbf{x}^*$. In particular, we replace $N(\mathbf{x}^*)$ with a set of points which are two units away from $\mathbf{x}^*$. Note that, from our numerical experience, it is not a good idea to initiate the minimization of the filled function too far from $\mathbf{x}^*$, though, as we are more likely to miss a point in a lower basin near to $\mathbf{x}^*$.

Algorithm 3.3 *Variation 1*

1. *Initialize* $\mathbf{x}_0 \in X$, ρ_0, μ_0, $\rho_L > 0$, $0 < \hat{\rho} < 1$, *and* $0 < \hat{\mu} < 1$.

   Let $\rho := \rho_0$ *and* $\mu := \mu_0$.

   Choose an initial point $\mathbf{x}_0 \in X$.

2. *Starting from* $\mathbf{x}_0$, *minimize* $f(\mathbf{x})$ *using Algorithm 3.1 to obtain a local minimizer* $\mathbf{x}^*$ *of* f.

3. *Define* $\bar{N}(\mathbf{x}^*) = \{\mathbf{w} \in X | \, \mathbf{w} = \mathbf{x}^* \pm 2\boldsymbol{e}_i : i = 1, 2, \ldots, n\} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q\}$, $q \leq 2n$.

4. *(a) Set* $\ell := 1$.

   (b) If $f(\mathbf{w}_\ell) < f(\mathbf{x}^*)$, *set* $\mathbf{x}_0 := \mathbf{w}_\ell$ *and go to Step 2. Otherwise, go to (c) below.*

   (c) Set $\ell := \ell + 1$. *If* $\ell \leq q$, *go to (b) above. Otherwise, set* $\ell := 1$ *and go to (d) below.*

   (d) Set the current switching point $\mathbf{x}_c := \mathbf{w}_\ell$.

5. *(a) If there exists a direction* $\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)$ *such that* $f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}^*)$, *then set* $\mathbf{x}_0 := \mathbf{x}_c + \mathbf{d}$ *and go to Step 2. Otherwise, go to (b) below.*

   (b) Let $\mathcal{D}_1 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : \; f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}_c)$ *and* $G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$.

   If $\mathcal{D}_1 \neq \emptyset$, *set* $\mathbf{d}^* := \arg\min_{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)} \{f(\mathbf{x}_c + \mathbf{d}) + G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$.

   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to (c) below.*

   (c) Let $\mathcal{D}_2 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : \; G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$.

   If $\mathcal{D}_2 \neq \emptyset$, *set* $\mathbf{d}^* := \arg\min_{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)} \{G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$.

   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to Step 6.*

6. *Let* $\acute{\mathbf{x}} = \mathbf{x}_c$ *be the local minimizer of* $G_{\mu,\rho,\mathbf{x}^*}$ *obtained from Step 5.*

   (a) If $\acute{\mathbf{x}} \in \tilde{X}$, *set* $\ell := \ell + 1$. *If* $\ell > q$, *go to Step 7. Otherwise, go to Step 4(d).*

   (b) If $\acute{\mathbf{x}} \notin \tilde{X}$, *reduce* μ *by setting* $\mu := \hat{\mu}\mu$ *and go to Step 5(b).*

7. *Reduce* ρ *by setting* $\rho := \hat{\rho}\rho$. *If* $\rho < \rho_L$, *terminate the algorithm. The current* $\mathbf{x}^*$ *is taken as a global minimizer of the problem. Otherwise, set* $\ell := 1$ *and go to Step 4(d).*

The minimization of Rosenbrock's function via Algorithm 3.3 leads to the results in Table 3.2. We found that a higher total number of function evaluations

Table 3.2: Results of Algorithm 3.3 - Rosenbrock's Function.

$\mathbf{x}_0$	E_f	E_G	R_E
$[-5,-5,-5,-5,-5]^\top$	1636	4252	0.52352
$[-4,-4,-4,-4,-4]^\top$	1635	4252	0.52320
$[-3,-3,-3,-3,-3]^\top$	1595	4252	0.51040
$[-2,-2,-2,-2,-2]^\top$	1554	4252	0.49728
$[-1,-1,-1,-1,-1]^\top$	1514	4252	0.48448
$[0,0,0,0,0]^\top$	1475	4252	0.47200
$[1,1,1,1,1]^\top$	1427	4181	0.45664
$[2,2,2,2,2]^\top$	1450	4181	0.46400
$[3,3,3,3,3]^\top$	1828	5254	0.58496
$[4,4,4,4,4]^\top$	1855	5254	0.59360
$[5,5,5,5,5]^\top$	1850	5254	0.59200

is needed before a global solution is attained, compared to the application of Algorithm 3.2. In hindsight, this is most likely due to not searching for the minimum of the filled function as thoroughly as in Algorithm 3.2, since the starting points do not cover the neighbourhood of $\mathbf{x}^*$ as effectively. Although Algorithm 3.3 requires fewer discrete filled function evaluations than Algorithm 3.2 (i.e. E_G is lower), this do not enhance the algorithm's overall efficiency.

On the other hand, we notice that Algorithm 3.3 outperforms the standard algorithm when minimizing Colville's function from all starting points. A summary of the computational results for this problem is shown in Table 3.3. The average of total number of original function evaluations is 1547.7, which is 7.8% lower than that for the standard algorithm. Clearly, depending on the 'shape' of the objective function, Algorithm 3.3 can result in improved efficiency by bypassing points in the immediate neighbourhood of $\mathbf{x}^*$.

We also considered a further variation of Algorithm 3.3 with the hope of searching for an improved point more efficiently in the region $\bar{N}(\mathbf{x}^*)$. Instead of using the first improved point found in $\bar{N}(\mathbf{x}^*)$ to continue the minimization of f, we test all points in $\bar{N}(\mathbf{x}^*)$ and choose the most improved point, assuming it actually exists. Interestingly, for each starting point used, this variation of the algorithm yielded the same results for both E_f and E_G values in minimizing Rosenbrock's

Table 3.3: Results of Algorithm 3.3 - Colville's Function.

$\mathbf{x}_0$	E_f	E_G	R_E
$[1,1,0,0]^\top$	1346	4972	0.006920985
$[1,1,1,1]^\top$	1329	4942	0.006833572
$[-10,10,-10,10]^\top$	2279	7254	0.011718368
$[-10,-5,0,5]^\top$	1492	5018	0.007671701
$[-10,0,0,-10]^\top$	1475	4972	0.007584288
$[0,0,0,0]^\top$	1365	4983	0.007018680

function, as shown in Table 3.2. It seems that none of the points in $\bar{N}(\mathbf{x}^*)$ is ever an improved point in the case of Rosenbrock's function, and this variation therefore yields no improvement over Algorithm 3.3. For Colville's function, we found this variation shows a similar results to those from Algorithm 3.3 itself, with an average $E_f = 1548.7$ compared to $E_f = 1547.7$ obtained in Algorithm 3.3. There appears to be no reason for pursuing this variation of Algorithm 3.3.

3.3 The Second Variation

Once again, we replace the set $N(\mathbf{x}^*)$ in Step 3 of the Algorithm 3.2, this time with a set of random points from X. Then, an additional step is added to test whether any one of these random points happens to be an improved point. The motivation for this algorithm is to search for improved points more efficiently by choosing points which give a broader coverage of X, similar to the methods proposed in [127, 128, 162].

Algorithm 3.4 *Variation 2*

1. *Initialize* $\mathbf{x}_0 \in X$, ρ_0, μ_0, $\rho_L > 0$, $0 < \hat{\rho} < 1$, *and* $0 < \hat{\mu} < 1$.

   Let $\rho := \rho_0$ *and* $\mu := \mu_0$.

   Choose an initial point $\mathbf{x}_0 \in X$.

2. *Starting from* $\mathbf{x}_0$, *minimize* $f(\mathbf{x})$ *using Algorithm 3.1 to obtain a local minimizer* $\mathbf{x}^*$ *of* f.

3. *Let* $M = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q\}$*, where* $\mathbf{w}_\ell,\ \ell = 1, \ldots, q$ *are randomly chosen from* X *and* $q = 2n$*.*

4. *(a) Set* $\ell := 1$*.*

   (b) If $f(\mathbf{w}_\ell) < f(\mathbf{x}^*)$*, set* $\mathbf{x}_0 := \mathbf{w}_\ell$ *and go to Step 2. Otherwise, go to (c) below. (c) Set* $\ell := \ell + 1$*. If* $\ell \leq q$*, go to (b) above. Otherwise, set* $\ell := 1$ *and go to (d) below.*

   (d) Set the current switching point $\mathbf{x}_c := \mathbf{w}_\ell$*.*

5. *(a) If there exists a direction* $\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)$ *such that* $f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}^*)$*, then set* $\mathbf{x}_0 := \mathbf{x}_c + \mathbf{d}$ *and go to Step 2. Otherwise, go to (b) below.*

   (b) Let $\mathcal{D}_1 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : \ f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}_c)$ *and* $G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$*.*

   If $\mathcal{D}_1 \neq \emptyset$*, set* $\mathbf{d}^* := \arg\min_{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)} \{f(\mathbf{x}_c + \mathbf{d}) + G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$*.*

   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to (c) below.*

   (c) Let $\mathcal{D}_2 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : \ G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$*.*

   If $\mathcal{D}_2 \neq \emptyset$*, set* $\mathbf{d}^* := \arg\min_{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)} \{G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$*.*

   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to Step 6.*

6. *Let* $\acute{\mathbf{x}} = \mathbf{x}_c$ *be the local minimizer of* $G_{\mu,\rho,\mathbf{x}^*}$ *obtained from Step 5.*

   (a) If $\acute{\mathbf{x}} \in \tilde{X}$*, set* $\ell := \ell + 1$*. If* $\ell > q$*, go to Step 7. Otherwise, go to Step 4(d).*

   (b) If $\acute{\mathbf{x}} \notin \tilde{X}$*, reduce* μ *by setting* $\mu := \hat{\mu}\mu$ *and go to Step 5(b).*

7. *Reduce* ρ *by setting* $\rho := \hat{\rho}\rho$*. If* $\rho < \rho_L$*, terminate the algorithm. The current* $\mathbf{x}^*$ *is taken as a global minimizer of the problem. Otherwise, set* $\ell := 1$ *and go to Step 4(d).*

Table 3.4 shows the numerical results of minimizing Rosenbrock's function using Algorithm 3.4 and also Algorithm 3.2 discussed earlier. Note that this algorithm requires far fewer of evaluations of both f and $G_{\mu,\rho,\mathbf{x}}$ when compared with

Table 3.4: Results of Algorithm 3.4 - Rosenbrock's Function.

$\mathbf{x}_0$	E_f	E_G	R_E	N_T
$[-5,-5,-5,-5,-5]^\top$	937	2115	0.29984	1
$[-4,-4,-4,-4,-4]^\top$	1219	2662	0.39008	1
$[-3,-3,-3,-3,-3]^\top$	1445	3279	0.46240	1
$[-2,-2,-2,-2,-2]^\top$	923	2060	0.29536	1
$[-1,-1,-1,-1,-1]^\top$	1032	2409	0.33024	2
$[0,0,0,0,0]^\top$	936	2447	0.29952	2
$[1,1,1,1,1]^\top$	820	1961	0.26240	1
$[2,2,2,2,2]^\top$	871	2165	0.27872	1
$[3,3,3,3,3]^\top$	1233	2904	0.39456	3
$[4,4,4,4,4]^\top$	1356	3112	0.43392	3
$[5,5,5,5,5]^\top$	980	2269	0.31360	1

Algorithm 3.3 discussed earlier. Specifically, when it works, Algorithm 3.4 succeeds in finding the global solution of the problem with an average $R_E = 0.341876$, compared with $R_E = 0.456931$ obtained by Algorithm 3.2. In other words, Algorithm 3.4 is able to minimize Rosenbrock's function much more efficiently than Algorithm 3.2, with a reduction of 25% in the total number of original function evaluations. However, for several starting points used in Algorithm 3.4, namely $[-1,-1,-1,-1,-1]^\top$, $[0,0,0,0,0]^\top$, $[3,3,3,3,3]^\top$, and $[4,4,4,4,4]^\top$, we were unable initially to determine the global solution for some choices of the random set M. In these cases, we repeated the application of the algorithm several times until the global optimum was obtained (note that the random set M changes with each new application). Note that the E_f values in Table 3.4 show the number of function evaluations recorded for the successful application of the algorithm only. The number of required attempts before reaching the global solution is denoted by N_T in Table 3.4.

Similarly, Algorithm 3.4 succeeds in determining the global solution of Colville's function much more efficiently with an average $E_f = 1143.2$, compared with $E_f = 1679.5$ obtained by Algorithm 3.2, which is a reduction of 31.9% in average total number of original function evaluations (see Table 3.14). Again, the gain in efficiency for Algorithm 3.4 is offset by reduced reliability, where we have to

Table 3.5: Results of Algorithm 3.4 - Colville's Function.

$\mathbf{x}_0$	E_f	E_G	R_E	N_T
$[1,1,0,0]^\top$	1092	2812	0.005614944	13
$[1,1,1,1]^\top$	1030	2387	0.005296147	1
$[-10,10,-10,10]^\top$	1106	2659	0.005686931	7
$[-10,-5,0,5]^\top$	1542	3759	0.007928795	15
$[-10,0,0,-10]^\top$	1135	3101	0.005836046	2
$[0,0,0,0]^\top$	954	3010	0.004905364	12

repeat the algorithm several times for each starting point before a global solution is attained, as shown in Table 3.5.

3.4 The Third Variation

We propose a similar algorithm to Algorithm 3.4 where the best improved random point is identified from set M in Step 4 to increase the computational efficiency. If such a point exists, the algorithm reverts to finding a better local minimizer of f in X.

Algorithm 3.5 *Variation 3*

1. *Initialize* $\mathbf{x}_0 \in X$, ρ_0, μ_0, $\rho_L > 0$, $0 < \hat{\rho} < 1$, *and* $0 < \hat{\mu} < 1$.
   Let $\rho := \rho_0$ *and* $\mu := \mu_0$.
   Choose an initial point $\mathbf{x}_0 \in X$.

2. *Starting from* $\mathbf{x}_0$, *minimize* $f(\mathbf{x})$ *using Algorithm 3.1 to obtain a local minimizer* $\mathbf{x}^*$ *of* f.

3. *Let* $M = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q\}$, *where* $\mathbf{w}_\ell,\ \ell = 1, \ldots, q$ *are randomly chosen from* X *and* $q = 2n$.

4. *(a) Let* $\mathbf{y} \in M$ *be such that* $f(\mathbf{y}) \leq f(\mathbf{w}_\ell),\ \ell = 1, \ldots, q$.
   If $f(\mathbf{y}) < f(\mathbf{x}^*)$, *set* $\mathbf{x}_0 := \mathbf{y}$ *and go to Step 2. Otherwise, set* $\ell := 1$ *and go*

to (b) below.

(b) Set the current switching point $\mathbf{x}_c := \mathbf{w}_\ell$.

5. *(a) If there exists a direction* $\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)$ *such that* $f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}^*)$*, then set* $\mathbf{x}_0 := \mathbf{x}_c + \mathbf{d}$ *and go to Step 2. Otherwise, go to (b) below.*

   (b) Let $\mathcal{D}_1 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : \ f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}_c)$ *and* $G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$.

   If $\mathcal{D}_1 \neq \emptyset$*, set* $\mathbf{d}^* := \arg\min_{\mathbf{d}\in\mathcal{D}(\mathbf{x}_c)}\{f(\mathbf{x}_c + \mathbf{d}) + G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$.

   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to (c) below.*

   (c) Let $\mathcal{D}_2 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : \ G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$.

   If $\mathcal{D}_2 \neq \emptyset$*, set* $\mathbf{d}^* := \arg\min_{\mathbf{d}\in\mathcal{D}(\mathbf{x}_c)}\{G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$.

   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to Step 6.*

6. *Let* $\acute{\mathbf{x}} = \mathbf{x}_c$ *be the local minimizer of* $G_{\mu,\rho,\mathbf{x}^*}$ *obtained from Step 5.*

   (a) If $\acute{\mathbf{x}} \in \tilde{X}$*, set* $\ell := \ell + 1$*. If* $\ell > q$*, go to Step 7. Otherwise, go to Step 4(b).*

   (b) If $\acute{\mathbf{x}} \notin \tilde{X}$*, reduce* μ *by setting* $\mu := \hat{\mu}\mu$ *and go to Step 5(b).*

7. *Reduce* ρ *by setting* $\rho := \hat{\rho}\rho$*. If* $\rho < \rho_L$*, terminate the algorithm. The current* $\mathbf{x}^*$ *is taken as a global minimizer of the problem. Otherwise, set* $\ell := 1$ *and go to Step 4(b).*

The results of minimizing Rosenbrock's function using Algorithm 3.5 are summarized in Table 3.6. Some starting points, such as $[-4,-4,-4,-4,-4]^\top$, $[-3,-3,-3,-3,-3]^\top$, $[0,0,0,0,0]^\top$, $[3,3,3,3,3]^\top$, and $[4,4,4,4,4]^\top$ show improvement over the function evaluations, compared with Algorithm 3.4. However, Algorithm 3.5 has only slightly lower average values of E_f and E_G as shown in Table 3.12. This indicates that the new Step 4 in Algorithm 3.5 fails to provide any improved point to increase the computational efficiency. Though some points converge to the global solution in fewer function evaluations, this is due to different set of random points being generated in the implemented algorithm. Similar to Algorithm 3.4, this algorithm succeeds in determining the global solution

Table 3.6: Results of Algorithm 3.5 - Rosenbrock's Function.

$\mathbf{x}_0$	E_f	E_G	R_E	N_T
$[-5,-5,-5,-5,-5]^\top$	1024	2144	0.32768	8
$[-4,-4,-4,-4,-4]^\top$	1006	2060	0.32192	1
$[-3,-3,-3,-3,-3]^\top$	1343	3140	0.42976	1
$[-2,-2,-2,-2,-2]^\top$	1050	2285	0.33600	1
$[-1,-1,-1,-1,-1]^\top$	1034	2409	0.33088	1
$[0,0,0,0,0]^\top$	923	2309	0.29536	1
$[1,1,1,1,1]^\top$	1313	3180	0.42016	1
$[2,2,2,2,2]^\top$	993	2175	0.31776	1
$[3,3,3,3,3]^\top$	888	2047	0.28416	1
$[4,4,4,4,4]^\top$	964	2284	0.30848	2
$[5,5,5,5,5]^\top$	1079	2649	0.34528	1

Table 3.7: Results of Algorithm 3.5 - Colville's Function.

$\mathbf{x}_0$	E_f	E_G	R_E	N_T
$[1,1,0,0]^\top$	1038	2607	0.005337282	6
$[1,1,1,1]^\top$	1054	2643	0.005419553	1
$[-10,10,-10,10]^\top$	977	2174	0.005023627	7
$[-10,-5,0,5]^\top$	1168	2607	0.006005728	2
$[-10,0,0,-10]^\top$	1168	2607	0.006005728	9
$[0,0,0,0]^\top$	1185	3175	0.006093140	5

much more efficiently than the standard algorithm, though it is less reliable. Note that the algorithm needs to be implemented more than once for starting points $[-5,-5,-5,-5,-5]^\top$ and $[4,4,4,4,4]^\top$ as the initial attempts fail to reach the global solution.

Table 3.7 summarizes the results of minimizing Colville's function using Algorithm 3.5. The outcomes show that Algorithm 3.5 also outperforms its predecessor where a lower E_f and E_G as are obtained. Besides, Algorithm 3.5 requires less attempts in attaining the global solution (see N_T values in Table 3.7) for most starting points, except $[-10,0,0,-10]^\top$. Note that Algorithm 3.5 shows 34.6% improvement over the standard algorithm, although it is less reliable.

3.5 The Fourth Variation

Although the idea of using the random points in Algorithm 3.4 has shown promising results as suggested in Table 3.4, its lack of reliability does not make it attractive for general problems. We propose another variation to overcome this issue, by combining Algorithms 3.3 and 3.4 discussed earlier. Firstly, a set of random points is tested to see if an improved point exists among them in Step 3. If none of these is an improved point, we then set up a set of points of $\bar{N}(\mathbf{x}^*)$ as outlined in Step 5 below. If no improved point is found in $\bar{N}(\mathbf{x}^*)$, we perform a local search of the filled function in Step 7.

Algorithm 3.6 *Variation 4*

1. *Initialize* $\mathbf{x}_0 \in X$, ρ_0, μ_0, $\rho_L > 0$, $0 < \hat{\rho} < 1$, *and* $0 < \hat{\mu} < 1$.
   Let $\rho := \rho_0$ *and* $\mu := \mu_0$.
   Choose an initial point $\mathbf{x}_0 \in X$.

2. *Starting from* $\mathbf{x}_0$, *minimize* $f(\mathbf{x})$ *using Algorithm 3.1 to obtain a local minimizer* $\mathbf{x}^*$ *of* f.

3. *Let* $M = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_p\}$, *where* $\mathbf{w}_j,\ j = 1, \ldots, p$ *are randomly chosen from* X *and* $p = 2n$.

4. *(a) Set* $j := 1$.
   (b) If $f(\mathbf{w}_j) < f(\mathbf{x}^*)$, *set* $\mathbf{x}_0 := \mathbf{w}_j$ *and go to Step 2. Otherwise, go to (c) below.*
   (c) Set $j := j + 1$. *If* $j \leq p$, *go to (b) above. Otherwise, go to Step 5.*

5. *Define* $\bar{N}(\mathbf{x}^*) = \{\mathbf{w} \in X | \, \mathbf{w} = \mathbf{x}^* \pm 2\boldsymbol{e}_i : i = 1, 2, \ldots, n\} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q\}$, $q \leq 2n$.

6. *(a) Set* $\ell := 1$.
   (b) If $f(\mathbf{w}_\ell) < f(\mathbf{x}^*)$, *set* $\mathbf{x}_0 := \mathbf{w}_\ell$ *and go to Step 2. Otherwise, go to (c)*

below.

(c) Set $\ell := \ell + 1$. *If* $\ell \leq q$, *go to (b) above. Otherwise, set* $\ell := 1$ *and go to (d) below.*

(d) Set the current switching point $\mathbf{x}_c := \mathbf{w}_\ell$.

7. *(a) If there exists a direction* $\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)$ *such that* $f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}^*)$, *then set* $\mathbf{x}_0 := \mathbf{x}_c + \mathbf{d}$ *and go to Step 2. Otherwise, go to (b) below.*

   (b) Let $\mathcal{D}_1 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}_c)$ *and* $G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$.

   If $\mathcal{D}_1 \neq \emptyset$, *set* $\mathbf{d}^* := \arg\min_{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)} \{f(\mathbf{x}_c + \mathbf{d}) + G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$.

   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to (c) below.*

   (c) Let $\mathcal{D}_2 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$.

   If $\mathcal{D}_2 \neq \emptyset$, *set* $\mathbf{d}^* := \arg\min_{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)} \{G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$.

   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to Step 8.*

8. *Let* $\acute{\mathbf{x}} = \mathbf{x}_c$ *be the local minimizer of* $G_{\mu,\rho,\mathbf{x}^*}$ *obtained from Step 7.*

   (a) If $\acute{\mathbf{x}} \in \tilde{X}$, *set* $\ell := \ell + 1$. *If* $\ell > q$, *go to Step 9. Otherwise, go to Step 6(d).*

   (b) If $\acute{\mathbf{x}} \notin \tilde{X}$, *reduce* μ *by setting* $\mu := \hat{\mu}\mu$ *and go to Step 7(b).*

9. *Reduce* ρ *by setting* $\rho := \hat{\rho}\rho$. *If* $\rho < \rho_L$, *terminate the algorithm. The current* $\mathbf{x}^*$ *is taken as a global minimizer of the problem. Otherwise, set* $\ell := 1$ *and go to Step 6(d).*

Unfortunately, Algorithm 3.6 results in a relatively high total number of original function evaluations compared with Algorithm 3.2, as displayed in Table 3.8. This may be due to the reason that more original function evaluations are needed to determine the global solution when the starting points are farther from $\mathbf{x}^*$. Interestingly, the E_f values obtained here are close to those obtained by Algorithm 3.3, except more function evaluations are recorded to evaluate ten random points in Step 4. In fact, both algorithms show the same E_G values for each starting point used in

Table 3.8: Results of Algorithm 3.6 - Rosenbrock's Function.

$\mathbf{x}_0$	E_f	E_G	R_E
$[-5,-5,-5,-5,-5]^\top$	1646	4252	0.52672
$[-4,-4,-4,-4,-4]^\top$	1645	4252	0.52640
$[-3,-3,-3,-3,-3]^\top$	1605	4252	0.51360
$[-2,-2,-2,-2,-2]^\top$	1564	4252	0.50048
$[-1,-1,-1,-1,-1]^\top$	1524	4252	0.48768
$[0,0,0,0,0]^\top$	1485	4252	0.47520
$[1,1,1,1,1]^\top$	1437	4181	0.45984
$[2,2,2,2,2]^\top$	1460	4181	0.46720
$[3,3,3,3,3]^\top$	1838	5254	0.58816
$[4,4,4,4,4]^\top$	1865	5254	0.59680
$[5,5,5,5,5]^\top$	1860	5254	0.59520

solving Rosenbrock's function. None of the random points appear to result in an improved point, thus giving similar results to those from Algorithm 3.3.

On the contrary when applied to Colville's function, Table 3.9 shows that Algorithm 3.6 yields a lower total number of original function evaluations compared with the standard algorithm. Still, since virtually none of the random points proposed in Step 3 ever yields an improved point, Algorithm 3.6 offers no effective improvement over Algorithm 3.3.

In addition, we also tested a further variation of Algorithm 3.6, where instead of looking for an improved point in $\bar{N}(\mathbf{x}^*)$, the proposed variation performs the local search of the filled function directly if no improved random point is identified in Step 4. Then, one of the points in $\bar{N}(\mathbf{x}^*)$ is used to initialize the minimization of the filled function. Though some starting points yield a lower number of function evaluations with this variation of Algorithm 3.6, its overall performance is similar to that of Algorithm 3.6 with the average total number of original function evaluations $E_f = 1625.45$ being relatively high compared with Algorithm 3.2. For Colville's function, the variation of Algorithm 3.6 shows a slightly improved performance with the average $E_f = 1548$ compared with an average $E_f = 1555.7$ obtained for Algorithm 3.6.

Table 3.9: Results of Algorithm 3.6 - Colville's Function.

$\mathbf{x}_0$	E_f	E_G	R_E
$[1,1,0,0]^\top$	1354	4972	0.006962120
$[1,1,1,1]^\top$	1337	4942	0.006874708
$[-10,10,-10,10]^\top$	2287	7254	0.011759503
$[-10,-5,0,5]^\top$	1500	5018	0.007712836
$[-10,0,0,-10]^\top$	1483	4972	0.007625424
$[0,0,0,0]^\top$	1373	4983	0.007059816

3.6 The Fifth Variation

Finally, we combine the random point concept with the standard algorithm outlined in the first section. An additional step to test if an improved random point exists is introduced before Step 3 in the standard algorithm.

Algorithm 3.7 *Variation 5*

1. *Initialize* $\mathbf{x}_0 \in X$, ρ_0, μ_0, $\rho_L > 0$, $0 < \hat{\rho} < 1$, *and* $0 < \hat{\mu} < 1$.

   Let $\rho := \rho_0$ *and* $\mu := \mu_0$.

   Choose an initial point $\mathbf{x}_0 \in X$.

2. *Starting from* $\mathbf{x}_0$, *minimize* $f(\mathbf{x})$ *using Algorithm 3.1 to obtain a local minimizer* $\mathbf{x}^*$ *of* f.

3. *Let* $M = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_p\}$, *where* $\mathbf{w}_j,\ j = 1, \ldots, p$ *are randomly chosen from* X *and* $p = 2n$.

4. *(a) Set* $j := 1$.

   (b) If $f(\mathbf{w}_j) < f(\mathbf{x}^*)$, *set* $\mathbf{x}_0 := \mathbf{w}_j$ *and go to Step 2. Otherwise, go to (c) below.*

   (c) Set $j := j + 1$. *If* $j \leq p$, *go to (b) above. Otherwise, go to Step 5.*

5. *(a) List the neighbouring points of* $\mathbf{x}^*$ *as* $N(\mathbf{x}^*) = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q\}$. *Set* $\ell := 1$.

(b) Set the current switching point, $\mathbf{x}_c := \mathbf{w}_\ell$.

6. *(a) If there exists a direction* $\mathbf{d} \in \mathcal{D}(\mathbf{x}_c)$ *such that* $f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}^*)$*, then set* $\mathbf{x}_0 := \mathbf{x}_c + \mathbf{d}$ *and go to Step 2. Otherwise, go to (b) below.*

   (b) Let $\mathcal{D}_1 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : \ f(\mathbf{x}_c + \mathbf{d}) < f(\mathbf{x}_c)$ *and* $G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$.

   If $\mathcal{D}_1 \neq \emptyset$*, set* $\mathbf{d}^* := \arg\min_{\mathbf{d}\in\mathcal{D}(\mathbf{x}_c)}\{f(\mathbf{x}_c + \mathbf{d}) + G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$.

   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to (c) below.*

   (c) Let $\mathcal{D}_2 = \{\mathbf{d} \in \mathcal{D}(\mathbf{x}_c) : \ G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c + \mathbf{d}) < G_{\mu,\rho,\mathbf{x}}(\mathbf{x}_c)\}$.

   If $\mathcal{D}_2 \neq \emptyset$*, set* $\mathbf{d}^* := \arg\min_{\mathbf{d}\in\mathcal{D}(\mathbf{x}_c)}\{G_{\mu,\rho,\mathbf{x}^*}(\mathbf{x}_c + \mathbf{d})\}$.

   Then, set $\mathbf{x}_c := \mathbf{x}_c + \mathbf{d}^*$ *and go to (a) above. Otherwise, go to Step 7.*

7. *Let* $\acute{\mathbf{x}} = \mathbf{x}_c$ *be the local minimizer of* $G_{\mu,\rho,\mathbf{x}^*}$ *obtained from Step 6.*

   (a) If $\acute{\mathbf{x}} \in \tilde{X}$*, set* $\ell := \ell + 1$*. If* $\ell > q$*, go to Step 8. Otherwise, go to Step 5(b).*

   (b) If $\acute{\mathbf{x}} \notin \tilde{X}$*, reduce* μ *by setting* $\mu := \hat{\mu}\mu$ *and go to Step 6(b).*

8. *Reduce* ρ *by setting* $\rho := \hat{\rho}\rho$*. If* $\rho < \rho_L$*, terminate the algorithm. The current* $\mathbf{x}^*$ *is taken as a global minimizer of the problem. Otherwise, set* $\ell := 1$ *and go to Step 5(b).*

The outcomes from Tables 3.10 and 3.12 show that Algorithm 3.7 is a better method compared to Algorithm 3.6 with a lower average total number of original function evaluations in minimizing Rosenbrock's function. Although Algorithm 3.6 performs the local search of the filled function more efficiently, this algorithm needs more function evaluations to reach the global solution.

Interestingly, Algorithm 3.7 seems to be the least efficient method among all algorithms tested for Colville's function, based on the results in Table 3.11. This may be due to the presence of many more local minima.

Table 3.10: Results of Algorithm 3.7 - Rosenbrock's Function.

$\mathbf{x}_0$	E_f	E_G	R_E
$[-5,-5,-5,-5,-5]^\top$	1446	4475	0.46272
$[-4,-4,-4,-4,-4]^\top$	1445	4475	0.46240
$[-3,-3,-3,-3,-3]^\top$	1405	4475	0.44960
$[-2,-2,-2,-2,-2]^\top$	1364	4475	0.43648
$[-1,-1,-1,-1,-1]^\top$	1324	4475	0.42368
$[0,0,0,0,0]^\top$	1284	4475	0.41088
$[1,1,1,1,1]^\top$	1262	4415	0.40384
$[2,2,2,2,2]^\top$	1281	4415	0.40992
$[3,3,3,3,3]^\top$	1657	5720	0.53024
$[4,4,4,4,4]^\top$	1676	5720	0.53632
$[5,5,5,5,5]^\top$	1674	5720	0.53568

Table 3.11: Results of Algorithm 3.7 - Colville's Function.

$\mathbf{x}_0$	E_f	E_G	R_E
$[1,1,0,0]^\top$	1434	5097	0.007373471
$[1,1,1,1]^\top$	1430	5076	0.007352903
$[-10,10,-10,10]^\top$	2682	5979	0.013790550
$[-10,-5,0,5]^\top$	1575	5134	0.008098477
$[-10,0,0,-10]^\top$	1565	5098	0.008047059
$[0,0,0,0]^\top$	1439	5099	0.007399180

Table 3.12: Comparison of Algorithms - Rosenbrock's Function, $n = 5$.

Types	$E_{f,avg}$	$E_{G,avg}$	$R_{E,avg}$
Algorithm 3.2	1427.909	4803.636	0.456931
Algorithm 3.3	1619.909	4512.364	0.518371
Algorithm 3.4	1068.364	2489.364	0.341876
Algorithm 3.5	1056.091	2425.636	0.337949
Algorithm 3.6	1629.909	4512.364	0.521571
Algorithm 3.7	1438	4803.636	0.460160

3.7 Concluding Remarks

From Tables 3.1-3.10, all discrete filled function algorithms eventually succeeded in finding the global minima of Rosenbrock's function and Colville's function from all starting points, although some required repeated starts. A summary of all computational results obtained from these algorithms for both problems are shown in Tables 3.12 and 3.14, respectively. Besides, we also tested all variations algorithms on a 25-dimensional Rossenbrock's function and summarized the outcomes in Table 3.13. The average total number of original function evaluations, the average total number of discrete filled function evaluations, and the average ratio of the average number of original function evaluations to reach the global solution to the total number of feasible points are denoted by $E_{f,avg}$, $E_{G,avg}$, and $R_{E,avg}$, respectively. For both problems, Algorithm 3.5 appears to be the most efficient algorithm with the least $R_{E,avg}$, although as noted, it is less reliable at actually being able to find the global solution. Interestingly, Algorithms 3.3-3.6 proposed here succeeded in minimizing Colville's function much more efficiently than the standard algorithm, but this was not the case for Rosenbrock's function. In view of these results, we choose to adopt Algorithm 3.2 directly for our work on discrete-valued optimal control problems in the later chapters.

Table 3.13: Comparison of Algorithms - Rosenbrock's Function, $n = 25$.

Types	$E_{f,avg}$	$E_{G,avg}$	$R_{E,avg}$
Algorithm 3.2	203125.8333	517281	1.87477×10^{-22}
Algorithm 3.3	271225	574238.1667	2.50330×10^{-22}
Algorithm 3.4	115697.1667	227910.8333	1.06784×10^{-22}
Algorithm 3.5	116686.3333	224788.8333	1.07697×10^{-22}
Algorithm 3.6	271290.6667	574238.1667	2.50390×10^{-22}
Algorithm 3.7	205656.6667	537205.1667	1.89813×10^{-22}

Table 3.14: Comparison of Algorithms - Colville's Function.

Types	$E_{f,avg}$	$E_{G,avg}$	$R_{E,avg}$
Algorithm 3.2	1679.5	5247.2	0.008635805
Algorithm 3.3	1547.7	5356.8	0.007957932
Algorithm 3.4	1143.2	2954.7	0.005878038
Algorithm 3.5	1098.3	2635.5	0.005647510
Algorithm 3.6	1555.7	5356.8	0.007999068
Algorithm 3.7	1687.5	5247.2	0.008676940

Chapter 4

Case Study: Hybrid Power System

This chapter proposes a new metaheuristic approach to optimize the operation of a hybrid power system. We first review the hybrid power system model and problem formulation reported in [116]. We then propose a new transformation, which converts the original problem into an equivalent mixed discrete optimization problem. Next, we outline a discrete filled function method and, based on this, develop a new metaheuristic algorithm to solve the problem at hand. Numerical results from the implementation of this algorithm are presented by the end of the chapter.

4.1 Hybrid Power System

A hybrid power system is a stand-alone electrical power system incorporating conventional (i.e. hydrocarbon powered) generators, renewable energy sources, and energy storage devices. Such systems are vital for electrification in remote areas, where grid-connected infrastructure is not available and fuel is expensive. Renewable energy sources, such as photovoltaic (PV) arrays, wind turbines, biomass, hydropower, and geothermal, are used to supplement the energy produced by the generators, thereby reducing fuel demand and maintenance costs. However, their contribution towards total energy output varies considerably throughout the day. For this reason, battery banks, and, in some cases, other storage devices such as hy-

drogen fuel cells, flywheels, and pumped water storage are used to store the excess energy generated from both conventional and renewable resources [91].

Apart from the start-up costs, the dominant running costs of a hybrid power system are associated with diesel generators and battery banks. The operating cost of a diesel generator is dependent on fuel consumption, maintenance costs, and loading. Frequent starts of the diesel generator from cold and running the generator for long hours at a low load increase engine wear and reduce fuel efficiency. On the other hand, incomplete charging and prolonged operation of a battery bank at a low charge state are two of the major factors limiting the battery bank life span. In fact, studies have shown that diesel generators and battery banks are likely to have significantly shortened lifetimes when operated under non-ideal conditions [91, 116, 150]. Hence, an efficient generator operating schedule is required to ensure a continuous electricity supply at the load, while at the same time keeping operating costs to a minimum.

This chapter proposes a new algorithm for determining an operating schedule that minimizes the total operating cost of a PV-diesel-battery hybrid power system. We adopt the model developed in [116], which is based on a hybrid power system consisting of a diesel generator as the main component, with a PV array providing additional energy and a battery bank for storage. The work in [116] concentrated on developing a mathematical model for hybrid power system operation and the application of a specialized optimal control technique to optimize the operation of the model. Further investigation has revealed that this optimization problem has many local minimizers.

4.2 Problem Formulation

4.2.1 A Discrete-Valued Control Problem

In this section, we briefly review the dynamic model of a hybrid power system discussed in [116]. Figure 4.1 illustrates the configuration of the hybrid power system

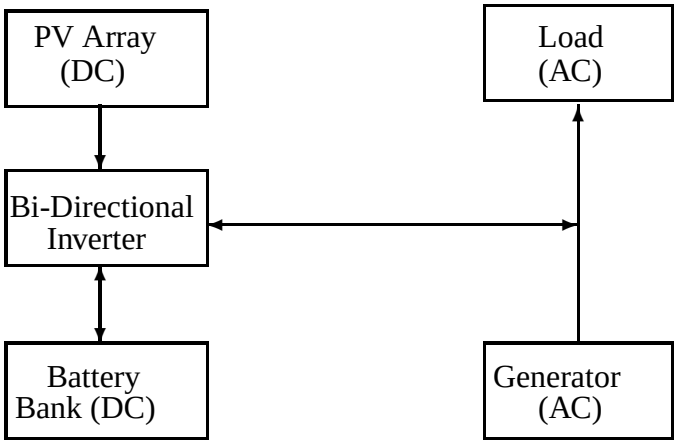

Figure 4.1: Schematic Diagram of a Hybrid Power System.

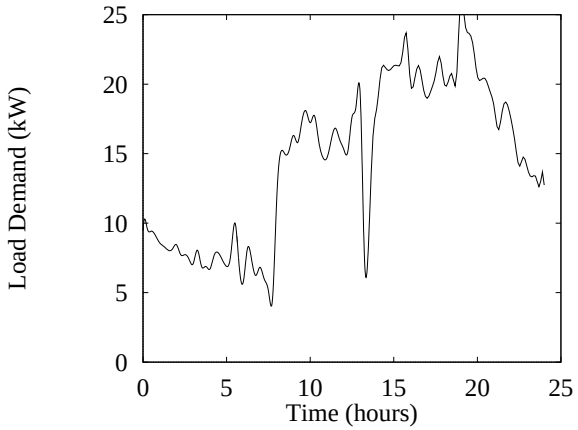

Figure 4.2: A Typical Load Demand Profile [116].

under consideration. It consists of an alternating current (AC) diesel generator, a bi-directional inverter, a PV array, and a battery bank for energy storage. The inverter is used to convert the direct current (DC) voltage of the PV array and the battery bank into AC, and vice versa. It also acts as the battery charger. The diesel generator is connected directly to the load to avoid conversion losses and thus increase the efficiency of the power system. The assumed load demand profile (see Figure 4.2) is based on data provided by the Centre for Renewable Energy & Sustainable Technologies Australia (CRESTA) [116]. The total daily load demand is approximately 340 kWh.

A battery bank of 100 kWh capacity is assumed here. Let $C(t)$ denote the capacity of the battery bank and $P_B(t)$ be the net power available at the battery

bank. The rate of charge of the battery bank is governed by

$$\dot{C}(t) = R(t) + D(t), \tag{4.1}$$

where the recharge rate is represented by

$$R(t) = \begin{cases} \dfrac{K_1 P_B(t)}{K_1 + C(t)}, & \text{if } P_B(t) \geq 0, \\ 0, & \text{if } P_B(t) < 0, \end{cases} \tag{4.2}$$

while the discharge rate is given by

$$D(t) = \begin{cases} K_2 P_B(t), & \text{if } P_B(t) < 0, \\ 0, & \text{if } P_B(t) \geq 0. \end{cases} \tag{4.3}$$

Note that $P_B(t) > 0$ indicates that the battery bank is undergoing charging while $P_B(t) < 0$ implies that the battery bank is being discharged. The parameters K_1 and K_2 assume the use of lead acid batteries, set up as 250 and 1.4, respectively. The parameters assume that the charging efficiency near full battery charge is just over 70% of the corresponding charging efficiency at a near empty battery state and 70% of power stored in the battery can be converted for load use, respectively.

We model the hybrid power system over the time horizon $[0, t_f]$, where t_f is the given terminal time. At each $t \in [0, t_f]$, there are three possible scenarios:

- The diesel generator is producing sufficient energy to meet the load demand and any excess power from the generator or PV array is directed to the battery bank.
- The power from the generator is insufficient to meet the load demand, so energy produced from the PV array is also used to supply the load. Any excess is directed to the battery bank.
- The combined power output from the diesel generator and PV array is insufficient to meet the load demand, so energy from the battery bank is required to make up the shortfall.

Then, on the basis of the above operating principles, the rate of change of the charge state is governed by the following dynamic equation:

$$\dot{C}(t) = \begin{cases} \dfrac{K_1 K_3 [P_R(t) + P_G(t) - P_L(t)]}{K_1 + C(t)}, & \text{if } P_G(t) \geq P_L(t), \\ \dfrac{K_1 [K_3 P_R(t) + P_G(t) - P_L(t)]}{K_1 + C(t)}, & \text{if } P_G(t) + K_3 P_R(t) \geq P_L(t), \\ K_2 \left[P_R(t) - \dfrac{P_L(t) - P_G(t)}{K_3} \right], & \text{if } P_G(t) + K_3 P_R(t) < P_L(t), \end{cases} \tag{4.4}$$

with

$$C(0) = C_0, \tag{4.5}$$

where C_0 is the given initial charge state, $P_R(t)$ is the power generated by the PV array at time t, $P_G(t)$ is the power produced by the diesel generator at time t, $P_L(t)$ is the load demand at time t, and K_1, K_2, K_3 are given model parameters. Both P_R and P_L are given functions derived from actual data supplied by CRESTA. On the other hand, P_G is the control function which is chosen by the system operator in practice.

Since the charge state must operate within a certain range, we have the following constraints:

$$C_{\min} \leq C(t) \leq C_{\max}, \quad \forall t \in [0, t_f], \tag{4.6}$$

and

$$C(t_f) = C_f, \tag{4.7}$$

where C_f is the desired final charge state, $C_{\min}$ and $C_{\max}$ are given constants.

Since it is difficult to continuously modify the power produced by the generator, we assume that the generator can only operate at certain fixed fractions of its

capacity. Suppose that there are M such levels. Then, we require

$$P_G(t) \in S = \{s_1, \ldots, s_M\}, \quad \forall t \in [0, t_f],$$

where, for each $i = 1, \ldots, M$, s_i denotes the power produced by the generator in mode i. According to [116], the operating cost of the diesel generator and the battery over the time horizon $[0, t_f]$ are given, respectively, by

$$\int_0^{t_f} P_G(t) g_1\left(\frac{100 P_G(t)}{P_{G,\max}}\right) dt$$

and

$$\int_0^{t_f} (C(t) - K_4)^2 dt,$$

where K_4 is a constant, $P_{G,\max}$ is the maximum power produced by the generator, and

$$g_1(x) = 2((0.2x + 0.5)^{0.4} - 0.5^{0.4})e^{-0.1x} + 0.15(1 - e^{-0.1x})$$

is a function derived from the data in [2]. The function g_1 is illustrated in Figure 4.3 and it reflects the fuel efficiency at different generated load levels. In practice, the problem is to choose the power produced by the generator so that these costs are minimized. This leads to the following optimal control problem.

Problem (A). Choose a discrete-valued control $P_G : [0, t_f] \to S$ such that the cost function

$$\alpha \int_0^{t_f} P_G(t) g_1\left(\frac{100 P_G(t)}{P_{G,\max}}\right) dt + \beta \int_0^{t_f} (C(t) - K_4)^2 dt$$

is minimized subject to the dynamics (4.4)-(4.5) and the constraints (4.6)-(4.7), where α and β are non-negative weights.

Problem (A) is a discrete-valued optimal control problem in which the control is restricted to take values in a discrete set. To determine the optimal discrete-valued control, we need to determine the order in which the different power levels are im-

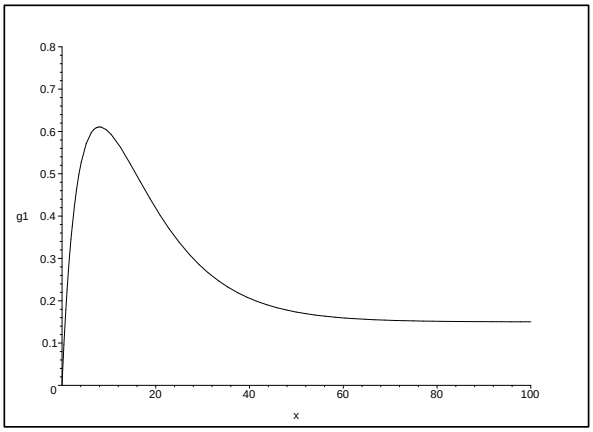

Figure 4.3: Profile of Function $g_1(x)$.

plemented (the switching sequence) and the times at which the power levels are changed (the switching times). However, conventional computational optimal control techniques are designed for problems in which the control takes values in a connected set, such as an interval, and hence they cannot solve Problem (A) directly. Moreover, variable switching times are known to cause problems in the implementations of any numerical algorithm [64, 70] for integrating the system dynamics. In the next subsection, we propose a new transformation to overcome these difficulties. This transformation introduces a new discrete variable to represent the switching sequence and a new continuous variable to represent the switching times. Using this transformation, we derive a new problem which is equivalent to Problem (A).

4.2.2 A Modified Time Scaling Transformation

Suppose that we allow the control to switch N times over the time horizon. Define a new time variable $\tau \in [0, N+1]$ with the partition $P_N = \{0, 1, 2, \ldots, N, N+1\}$. For each $i = 1, \ldots, N+1$, let

$$v_i \in \{1, \ldots, M\}$$

be a discrete variable representing the mode of the generator during the i-th subinterval. Let

$$\mathbf{v} = [v_1, \ldots, v_{N+1}]^T$$

and V be the set of all such vectors. For each $i = 1, \ldots, N+1$, we define a new control function $U_G(\tau, \mathbf{v})$ by

$$U_G(\tau, \mathbf{v}) = s_{v_i}, \qquad \tau \in [i-1, i).$$

Hence, $U_G(\tau, \mathbf{v})$ represents $P_G(t)$ in the new time scale and $U_{G,\max}$ represents the maximum generator capacity. Next, $u(\tau)$, the time scaling control, is defined as a piecewise constant function with possible discontinuities at $1, 2, \ldots, N$ and satisfying

$$0 \leq u(\tau) \leq t_f, \qquad \tau \in [0, N+1]. \tag{4.8}$$

Let U denote the class of all valid time scaling controls satisfying (4.8). The original time horizon $[0, t_f]$ is transformed into the new time horizon $[0, N+1]$ through the differential equation

$$\dot{t}(\tau) = u(\tau) \tag{4.9}$$

with

$$t(0) = 0, \tag{4.10}$$

and with the additional constraint

$$t(N+1) = t_f. \tag{4.11}$$

Therefore, the original dynamics (4.4)-(4.5) are transformed into

$$\dot{\bar{C}}(\tau) = \begin{cases} \dfrac{K_1 K_3[P_R(t(\tau)) + U_G(\tau, \mathbf{v}) - P_L(t(\tau))]}{K_1 + \bar{C}(\tau)} u(\tau), \\ \qquad \text{if } U_G(\tau, \mathbf{v}) \geq P_L(t(\tau)), \\ \dfrac{K_1[K_3 P_R(t(\tau)) + U_G(\tau, \mathbf{v}) - P_L(t(\tau))]}{K_1 + \bar{C}(\tau)} u(\tau), \\ \qquad \text{if } U_G(\tau, \mathbf{v}) + K_3 P_R(t(\tau)) \geq P_L(t(\tau)) > U_G(\tau, \mathbf{v}), \\ K_2 \left[P_R(t(\tau)) - \dfrac{P_L(t(\tau)) - U_G(\tau, \mathbf{v})}{K_3} \right] u(\tau), \\ \qquad \text{if } U_G(\tau, \mathbf{v}) + K_3 P_R(t(\tau)) < P_L(t(\tau)), \end{cases} \tag{4.12}$$

and

$$\bar{C}(0) = C_0. \tag{4.13}$$

Similarly, constraints (4.6) and (4.7) are transformed into

$$C_{\min} \leq \bar{C}(\tau) \leq C_{\max}, \quad \forall \tau \in [0, N+1], \tag{4.14}$$

and

$$\bar{C}(N+1) = C_f. \tag{4.15}$$

After the transformation, the terms measuring the fuel cost and the operating cost of the battery are

$$\int_0^{N+1} U_G(\tau, \mathbf{v}) g_1 \left(\frac{100 U_G(\tau, \mathbf{v})}{U_{G,\max}} \right) u(\tau) d\tau$$

and

$$\int_0^{N+1} (\bar{C}(\tau) - K_4)^2 u(\tau) d\tau,$$

respectively. On the basis of the above discussion, we have the following problem, which is equivalent to Problem (A).

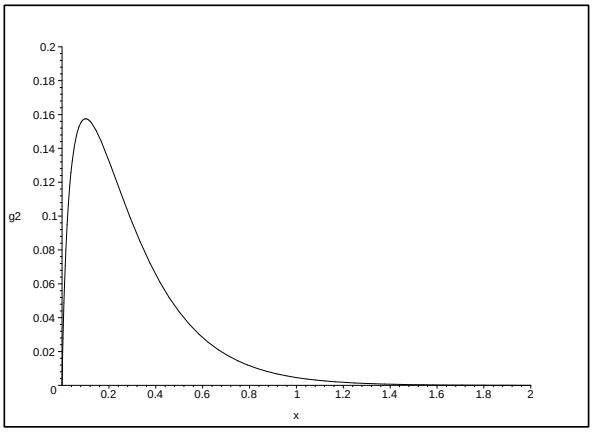

Figure 4.4: Profile of Function $g_2(x)$.

Problem (B). Choose $\mathbf{v} \in V$ and $u \in U$ such that the cost function

$$g_0(\mathbf{v}, u) = \alpha \int_0^{N+1} U_G(\tau, \mathbf{v}) g_1\left(\frac{100 U_G(\tau, \mathbf{v})}{U_{G,\max}}\right) u(\tau) d\tau + \beta \int_0^{N+1} (\bar{C}(\tau) - K_4)^2 u(\tau) d\tau$$

is minimized subject to the dynamics (4.9)-(4.13) and the constraints (4.14)-(4.15), where α and β are non-negative weights.

4.2.3 Penalizing Frequent Switching

Frequent switching is undesirable in practice because it significantly increases mechanical wear. However, there is no mechanism in Problem (B) to discourage a control schedule that frequently switches between generator modes. Hence, we would also like to minimize the term

$$\int_0^{N+1} g_2(u(\tau)) d\tau,$$

where

$$g_2(x) = ((x + 0.01)^{0.25} - 0.01^{0.25}) e^{-5x}.$$

The function g_2 is illustrated in Figure 4.4. This term severely penalizes an operating schedule that runs any generator mode for less than 15 minutes. Our new

problem is stated below.

Problem (C). Choose $\mathbf{v} \in V$ and $u \in U$ such that the cost function

$$g_0(\mathbf{v}, u) = \int_0^{N+1} \left\{ \alpha U_G(\tau, \mathbf{v}) g_1\left(\frac{100 U_G(\tau, \mathbf{v})}{U_{G,\max}}\right) u(\tau) + \beta(\bar{C}(\tau) - K_4)^2 u(\tau) + \gamma g_2(u(\tau)) \right\} d\tau \quad (4.16)$$

is minimized subject to dynamics (4.9)-(4.13) and the constraints (4.14)-(4.15), where $\alpha,\ \beta$, and γ are non-negative weights.

Note that Problem (C) is a mixed discrete dynamic optimization problem. Note further that the term penalizing frequent switching is also used in [116], where an alternative time scale transformation was employed. To facilitate the application of a global optimization technique, we decompose it into a bi-level optimization problem in the next subsection, where the upper level problem is a discrete optimization problem and the lower level problem is a conventional optimal control problem.

4.2.4 Decomposition of Problem (C)

In our numerical experiments, we have observed that multiple locally optimal solutions are found when different initial switching times are used to solve the model developed in [116] using the transformation and solution technique suggested there. Note that many practical discrete-valued optimal control problems exhibit similar behavior. Thus, with the transformation leading to Problem (C), we intend to apply the discrete filled function method in an attempt to determine a global optimal solution. For this purpose, we restructure Problem (C) by decomposing it into a bi-level optimization problem as follows.

Problem $(\mathbf{C_1})$. Given $\mathbf{v} \in V$, choose a $u \in U$ such that the cost function

$$g_0(u|\mathbf{v}) = \int_0^{N+1} \left\{ \alpha U_G(\tau, \mathbf{v}) g_1\left(\frac{100 U_G(\tau, \mathbf{v})}{U_{G,\max}}\right) u(\tau) + \beta(\bar{C}(\tau) - K_4)^2 u(\tau) + \gamma g_2(u(\tau)) \right\} d\tau \quad (4.17)$$

is minimized subject to the dynamics (4.9)-(4.13) and the constraints (4.14)-(4.15), where $\alpha,\ \beta$, and γ are non-negative weights.

Problem (C_1) is essentially a lower level problem or subproblem. It is simply a standard optimal control problem where the optimal value of g_0 in (4.17) can be determined using an optimal control software based on the concept of control parameterization, such as MISER3.3. The second problem in the decomposition is defined as follows.

Problem $(\mathbf{C_2})$. Choose $\mathbf{v} \in V$ such that the cost function

$$J(\mathbf{v}) \quad (4.18)$$

is minimized, where

$$J(\mathbf{v}) = \min_{u \in U} \quad g_0(\mathbf{v}, u).$$

Problem (C_2) represents the upper level of Problem (C). Clearly, Problem (C_2) is a purely discrete optimization problem. To compute the value of the objective function at $\mathbf{v} \in V$, we solve the subproblem (C_1) corresponding to $\mathbf{v} \in V$ using MISER3.3. Next, we propose a combined algorithm where Problem (C_2) will be solved using the discrete filled function method in Section 3.1, i.e. Algorithm 3.2, to determine a global solution and subproblem (C_1) is solved with MISER3.3. For our numerical computations, we have been able to incorporate the discrete filled function method into the MISER3.3 software. The details of the numerical results are discussed in the next section. Note that we set $\rho_L = 0.001$ for our numerical

computation to confirm a near global solution is attained from the algorithm.

To increase the efficiency, we construct a look-up table to store each value of the objective function J computed so far. Thus, we avoid repeated application of the subproblem solution algorithm at the same point. This is vital to the computational efficiency because computing $J(\mathbf{v})$ involves solving a complex optimal control problem, which takes considerable computational time. Note that for some sequences, the subproblem solution algorithm may report that Problem (C_1) is infeasible. This may be due to the subproblem solver (MISER3.3) not converging properly (the subproblem is somewhat ill-conditioned) or it may actually indicate that the subproblem is infeasible at the current sequence $\mathbf{v}$. In an effort to distinguish between these two possibilities, we re-initialize the optimization of the subproblem several times. When five such attempts fail to yield a feasible solution, it is assumed that no feasible solution of the subproblem exists for this switching sequence $\mathbf{v}$. An artificially high cost is assigned to such a sequence and the algorithm is allowed to continue.

4.3 Numerical Results

In this section, our algorithm is applied to solve Problem (C) with 4, 7, and 9 switches. A comparison between our method and the method in [116] is discussed at the end of this section. The results were computed using a modified version of MISER3.3 so that the filled function method is able to call on the standard MISER3.3 algorithm. The experiments were conducted on a Windows-based PC, with a CPU speed of 2.4GHz and 2GB RAM.

4.3.1 Results for 4 Switches

By setting $N = 4,\ k_1 = 250,\ k_2 = 1.4,\ k_3 = 0.9,\ k_4 = 80,\ C_0 = 80\,\text{kWh},\ C_{\min} = 20\,\text{kWh},\ C_{\max} = 100\,\text{kWh},\ \alpha = 1,\ \beta = 0.01,\ \gamma = 10,\ t_f = 24,\ c = 0.5,\ \mu_0 = 0.1,\ \rho_0 = 0.1,\ \omega = 1,\ \rho_L = 0.001,\ \hat{\rho} = 0.1,\ \hat{\mu} = 0.1$, we solved the corresponding

Problem (C). There are 3125 potential switching sequences for 4 switches with $U_G \in \{0, 8, 12, 16, 20\}$.

We tested the problem with 10 random initial sequences, namely, $[2, 3, 4, 5, 4]^T$, $[4, 5, 3, 5, 2]^T$, $[5, 1, 5, 1, 5]^T$, $[4, 3, 1, 5, 2]^T$, $[3, 4, 4, 3, 5]^T$, $[2, 4, 5, 4, 4,]^T$, $[2, 4, 5, 4, 1]^T$,$[5, 4, 3, 2, 3]^T$, $[2, 3, 2, 4, 5]^T$, and $[4, 5, 3, 4, 5]^T$. We found 13 local minimizers during the application of the algorithm on these ten starting points. For each starting sequence, the algorithm successfully identified the assumed discrete global minimizer, $[2, 3, 4, 5, 4]^T$, for which the cost function value is $J = 58.7216005$, and the time scaling control is

$$u = \begin{cases} 7.50650, \ 0 \leq \tau < 1, \\ 1.42257, \ 1 \leq \tau < 2, \\ 5.05190, \ 2 \leq \tau < 3, \\ 8.70635, \ 3 \leq \tau < 4, \\ 1.31267, \ 4 \leq \tau < 5. \end{cases}$$

Table 4.1 illustrates the computational results of 10 experiments which use the same initial time scaling control set at

$$u = \begin{cases} 1, \ 0 \leq \tau < 1, \\ 6, \ 1 \leq \tau < 2, \\ 8, \ 2 \leq \tau < 3, \\ 6, \ 3 \leq \tau < 4, \\ 3, \ 4 \leq \tau < 5. \end{cases}$$

The number of original function evaluations and filled function evaluations are denoted by E_J and E_G, respectively. Note that E_J does not include function evaluations that were obtained from the look-up table.

The algorithm terminates when $\mu = 1 \times 10^{-41}$ and $\rho = 1 \times 10^{-3}$, at which point no further improvement can be made. Therefore, $[2, 3, 4, 5, 4]^T$ is assumed to be the globally optimal sequence for Problem (C) with $N = 4$. At most, 571 switching sequences are computed during the ten applications of the algorithm, which is 18.3% of the total possible sequences. Further experiments with a range of refined

Table 4.1: Numerical Results for Problem (C) with 4 Switches.

$\mathbf{v_0}$	$\mathbf{v}^*$	J	E_J	E_G
$[4,5,3,5,2]^T$	$[5,5,2,5,4]^T$	6.35559358×10^1		
	$[3,5,2,5,4]^T$	6.35559357×10^1		
	$[5,2,4,5,4]^T$	5.88517382×10^1		
	$[3,2,4,5,4]^T$	5.87626319×10^1		
	$[2,3,4,5,4]^T$	5.87216005×10^1	350	1443
$[5,1,5,1,5]^T$	$[5,2,4,3,5]^T$	6.03626516×10^1		
	$[3,2,4,3,5]^T$	6.02617585×10^1		
	$[2,3,4,3,5]^T$	6.01837944×10^1		
	$[2,4,3,2,5]^T$	6.01491200×10^1		
	$[2,4,5,3,4]^T$	5.88517553×10^1		
	$[2,4,5,4,2]^T$	5.88517431×10^1		
	$[2,3,4,5,4]^T$	5.87216005×10^1	571	2636
$[4,3,1,5,2]^T$	$[3,2,4,5,4]^T$	5.87626319×10^1		
	$[2,3,4,5,4]^T$	5.87216005×10^1	336	1318
$[3,4,4,3,5]^T$	$[2,3,4,3,5]^T$	6.01837944×10^1		
	$[2,4,3,2,5]^T$	6.01491200×10^1		
	$[2,4,5,3,4]^T$	5.88517553×10^1		
	$[2,4,5,4,2]^T$	5.88517431×10^1		
	$[2,3,4,5,4]^T$	5.87216005×10^1	420	1830
$[2,4,5,4,4]^T$	$[2,4,5,4,2]^T$	5.88517431×10^1		
	$[2,3,4,5,4]^T$	5.87216005×10^1	362	1543
$[2,4,5,4,1]^T$	$[2,4,5,4,2]^T$	5.88517431×10^1		
	$[2,3,4,5,4]^T$	5.87216005×10^1	362	1543
$[5,4,3,2,3]^T$	$[2,4,5,4,2]^T$	5.88517431×10^1		
	$[2,3,4,5,4]^T$	5.87216005×10^1	363	1614
$[2,3,2,4,5]^T$	$[2,4,3,5,5]^T$	6.01491301×10^1		
	$[2,4,5,4,2]^T$	5.88517431×10^1		
	$[2,3,4,5,4]^T$	5.87216005×10^1	406	1750
$[4,5,3,4,5]^T$	$[4,5,2,4,5]^T$	6.03626306×10^1		
	$[2,3,4,5,4]^T$	5.87216005×10^1	386	1568
$[2,3,4,5,4]^T$	$[2,3,4,5,4]^T$	5.87216005×10^1	319	1220

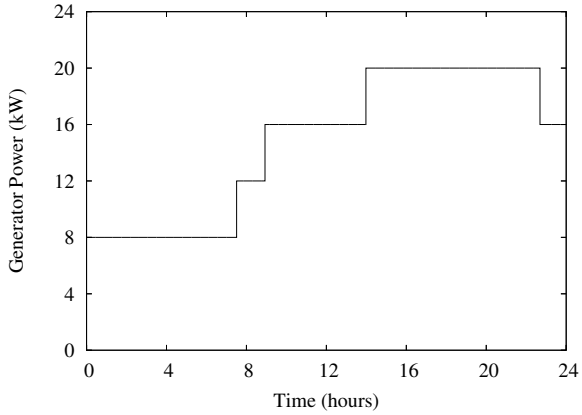

Figure 4.5: Optimal Generator Power Profile for 4 Switches.

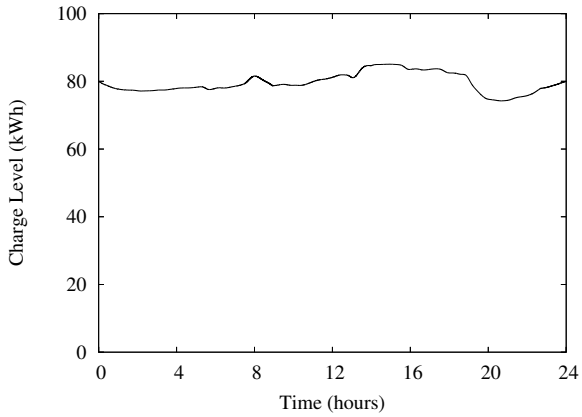

Figure 4.6: Optimal Battery Charge Profile for 4 Switches.

parameter values of the discrete filled function were carried out and the results also confirmed $[2, 3, 4, 5, 4]^T$ as the best solution.

From Table 4.1, most of the local minimizers start with 8 kW, and the generator needs to run at an average of 12 kW to achieve an optimal cost, based on the load demand and PV data. Figure 4.5 depicts the best operating strategy for the diesel generator: starts at a lower load, which is 8 kW for 7.5 hours, increase this to 12 kW for another 1.5 hours until it reaches maximum power at 20 kW, before reducing it to 16 kW. Note that the generator is maintained at a minimum of 12 kW for almost two thirds of the day (16.5 hours) to achieve its best performance. Figure 4.6 shows that the charge level of the battery bank remains almost constant for the first eight hours, before fluctuating between 75 kWh and 85 kWh for the rest of the day.

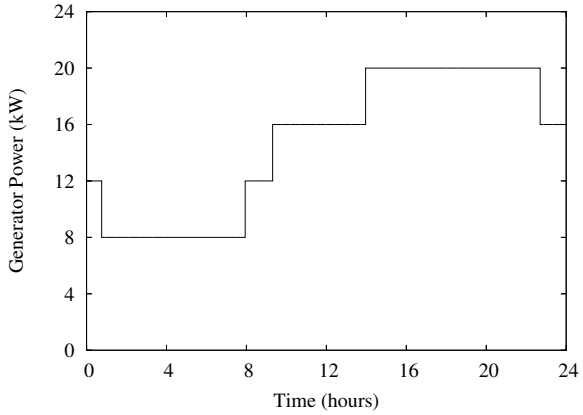

Figure 4.7: Optimal Generator Power Profile for 7 Switches.

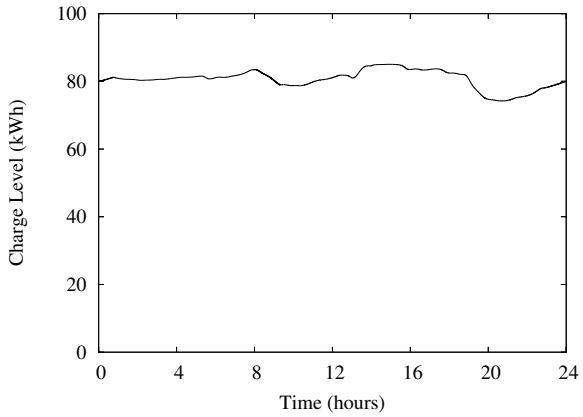

Figure 4.8: Optimal Battery Charge Profile for 7 Switches.

4.3.2 Results for 7 Switches

We apply the same algorithm to find the global switching sequence of Problem (C) for $N = 7$ switches. By using $u(\tau) = 3,\ \tau \in [0, 8]$, for 5 experiments, Table 4.2 indicates that $[4, 3, 2, 3, 4, 5, 5, 4]^T$ is likely to be the global minimizer as it resulted from using 5 different initial sequences. Thirty local minimizers were found with the proposed algorithm. Indeed, $[3, 2, 3, 4, 5, 4]^T$ is actually the optimal switching sequence for 7 switches when we take into account that the optimal u is zero over one interval of its defining partition. The optimal solution is 58.5886863, an improvement of 0.23% compared with the $N = 4$ case. The algorithm terminates when $\mu = 1 \times 10^{-5}$ and $\rho = 1 \times 10^{-2}$.

The plots of the generator output and battery charge level are shown in Fig-

Table 4.2: Numerical Results for Problem (C) with 7 Switches.

$\mathbf{v_0}$	$\mathbf{v}^*$	J	E_J	E_G
$[3,4,4,3,5,2,3,4]^T$	$[2,3,4,3,5,2,3,4]^T$	6.00318935×10^1		
	$[2,3,4,3,5,2,5,4]^T$	5.87216219×10^1		
	$[2,3,5,3,4,3,5,4]^T$	5.87215981×10^1		
	$[3,4,2,3,4,5,5,4]^T$	5.85886904×10^1		
	$[4,3,2,3,4,5,5,4]^T$	5.85886863×10^1	1733	3825
$[2,3,4,5,2,3,4,5]^T$	$[2,3,4,5,2,3,4,5]^T$	6.01837798×10^1		
	$[2,4,4,5,2,5,4,5]^T$	5.88519036×10^1		
	$[2,4,5,4,2,5,4,5]^T$	5.88518410×10^1		
	$[2,4,4,2,4,5,4,5]^T$	5.88517428×10^1		
	$[3,4,3,2,4,5,4,5]^T$	5.87628543×10^1		
	$[3,5,2,2,4,5,4,5]^T$	5.87626332×10^1		
	$[2,3,5,2,4,5,4,5]^T$	5.87216048×10^1		
	$[4,2,2,3,4,5,4,5]^T$	5.87215986×10^1		
	$[5,2,2,3,4,5,5,4]^T$	5.87215886×10^1		
	$[4,3,2,3,4,5,5,4]^T$	5.85886863×10^1	2550	5717
$[3,4,5,2,3,4,5,2]^T$	$[3,5,5,2,3,4,5,4]^T$	5.87216035×10^1		
	$[5,4,2,3,4,3,5,4]^T$	5.87216030×10^1		
	$[3,4,2,3,4,5,5,4]^T$	5.85886904×10^1		
	$[4,3,2,3,4,5,5,4]^T$	5.85886863×10^1	1659	3605
$[4,5,3,5,2,1,2,3]^T$	$[4,5,2,5,1,1,2,3]^T$	6.53276638×10^1		
	$[5,5,2,5,2,2,3,3]^T$	6.38284739×10^1		
	$[5,3,2,5,4,1,3,2]^T$	6.29162001×10^1		
	$[3,4,2,5,4,5,4,3]^T$	5.92223690×10^1		
	$[4,4,2,3,4,5,4,3]^T$	5.87216018×10^1		
	$[5,3,2,3,4,5,4,2]^T$	5.87215951×10^1		
	$[4,3,2,3,4,5,5,4]^T$	5.85886863×10^1	3115	5923
$[2,1,4,1,5,5,1,3]^T$	$[3,2,4,1,5,5,2,2]^T$	6.01135927×10^1		
	$[3,2,4,3,1,5,4,5]^T$	5.87626426×10^1		
	$[2,3,4,3,1,5,4,5]^T$	5.87216039×10^1		
	$[2,3,4,2,3,5,4,5]^T$	5.87216007×10^1		
	$[2,3,4,2,5,5,4,5]^T$	5.87215952×10^1		
	$[2,3,4,2,5,4,2,5]^T$	5.87215895×10^1		
	$[3,2,5,1,4,5,5,3]^T$	5.87123116×10^1		
	$[3,2,4,3,4,4,5,4]^T$	5.85888513×10^1		
	$[4,3,2,3,4,5,5,4]^T$	5.85886863×10^1	6233	11887

ures 4.7 and 4.8, respectively. Figure 4.7 shows that the generator should run at 12 kW for the first 42 minutes before following the profile of the solution in Figures 4.5. No significant differences are observed for the battery bank profiles between the tests with 4 switches and 7 switches. The computational results indicate that the filled function algorithm is robust and efficient in solving a large scale problem of up to 390,625 potential sequences, with less than 1.6% of the potential switching sequences computed during the search for the global optimum.

4.3.3 Results for 9 Switches

Table 4.3 depicts the numerical results of solving Problem (C) with $N = 9$ switches, which leads to 9,765,625 possible sequences. Five experiments were carried out using $u(\tau) = 2.4,\ \tau \in [0, 10]$, as the initial guess. Only 0.11% of all potential switching sequences are computed and the algorithm identifies 40 local minimizers. However, the algorithm fails to identify a unique global minimizer of Problem (C) in this case, and objective function values in the range from 58.2438575 to 58.55886706 are generated. The best solution from Table 4.3 is 58.2438575, which is an improvement over the solutions with 4 and 7 switches, by 0.81% and 0.59%, respectively. Clearly, as expected, better solutions are obtained when the number of switches is increased. However, the algorithm appears unable to consistently yield a global solution. This is probably because it cannot guarantee a globally optimal solution of the subproblems.

The characteristics of the generator and battery charge level for the best solution found are plotted in Figures 4.9 and 4.10, respectively. In contrast with Figures 4.5 and 4.7, where the generator is left running non-stop for 24 hours, Figure 4.9 shows that it is favorable to turn off the generator for 48 minutes early in the morning to avoid excess energy waste, before re-starting it at 16 kW near 8 am, and increase the generator to maximum capacity at 2 pm. The suggested operating strategy here is $[3, 2, 1, 4, 5, 4]^T$ (once again, the optimal u was zero over several subintervals of its defining partition).

Table 4.3: Numerical Results for Problem (C) with 9 Switches.

$\mathbf{v_0}$	$\mathbf{v}^*$	J	E_J	E_G
$[5,1,5,1,5,1,5,1,5,1]^T$	$[5,1,4,1,5,1,5,1,5,1]^T$	7.41514672×10^1		
	$[5,2,4,2,5,1,4,1,5,1]^T$	6.03626730×10^1		
	$[5,4,4,2,5,1,4,1,5,2]^T$	6.01994522×10^1		
	$[4,4,4,2,5,1,4,3,5,3]^T$	5.92987984×10^1		
	$[4,5,4,2,4,2,4,3,5,4]^T$	5.88517493×10^1		
	$[5,4,5,2,4,2,4,3,5,4]^T$	5.88517395×10^1		
	$[4,5,3,2,4,3,4,3,5,4]^T$	5.87626361×10^1		
	$[5,4,3,2,4,3,4,3,5,4]^T$	5.87626283×10^1		
	$[2,5,3,2,4,3,4,5,5,4]^T$	5.87284007×10^1		
	$[3,4,3,2,4,3,4,5,5,4]^T$	5.85886872×10^1		
	$[4,4,3,2,3,4,5,4,5,4]^T$	5.85886768×10^1		
	$[4,3,4,2,3,4,5,4,5,4]^T$	5.85886703×10^1		
	$[5,3,5,2,1,4,5,5,4,3]^T$	5.82438612×10^1	9398	16548
$[3,4,3,4,3,4,3,4,3,4]^T$	$[2,4,5,4,3,4,3,3,2,4]^T$	5.88517547×10^1		
	$[2,4,5,4,3,4,3,3,4,3]^T$	5.88517367×10^1		
	$[2,3,5,4,5,4,1,3,4,1]^T$	5.87216011×10^1		
	$[2,5,3,4,5,4,2,3,3,1]^T$	5.87216006×10^1		
	$[5,2,3,4,5,4,4,2,2,3]^T$	5.87215994×10^1		
	$[5,2,3,4,5,4,4,2,2,5]^T$	5.87215930×10^1		
	$[5,2,3,4,5,4,4,2,4,4]^T$	5.87215911×10^1		
	$[2,3,1,4,5,4,4,1,4,4]^T$	5.82784943×10^1	9380	16177
$[5,4,3,2,5,4,3,2,5,2]^T$	$[5,3,3,2,3,4,3,2,5,4]^T$	5.87215987×10^1		
	$[5,5,2,2,3,4,3,2,5,4]^T$	5.87215868×10^1		
	$[3,4,4,2,3,4,5,2,5,4]^T$	5.85886835×10^1		
	$[4,3,5,2,3,4,5,2,5,4]^T$	5.85886720×10^1		
	$[4,3,4,2,3,4,5,4,5,4]^T$	5.85886703×10^1	6674	11592
$[2,3,4,5,2,3,4,5,2,3]^T$	$[2,3,4,5,3,4,4,5,2,3]^T$	5.91803219×10^1		
	$[2,3,4,5,3,5,5,4,2,3]^T$	5.87225217×10^1		
	$[2,3,4,4,3,5,5,4,2,2]^T$	5.87216606×10^1		
	$[3,5,4,2,3,4,5,4,3,5]^T$	5.85886787×10^1		
	$[4,4,3,2,3,4,5,4,3,5]^T$	5.85886734×10^1		
	$[4,4,3,2,3,4,5,4,3,2]^T$	5.85886723×10^1		
	$[5,5,3,2,1,4,5,4,3,2]^T$	5.82438626×10^1		
	$[5,4,3,2,1,4,5,4,2,3]^T$	5.82438575×10^1	11047	18844
$[2,3,4,5,1,2,3,4,5,2]^T$	$[2,3,4,3,2,3,2,4,5,3]^T$	5.91801836×10^1		
	$[3,2,4,4,3,2,1,4,5,4]^T$	5.87626301×10^1		
	$[2,3,4,4,3,2,1,4,5,4]^T$	5.87216014×10^1		
	$[2,3,4,4,3,4,1,4,5,4]^T$	5.87216012×10^1		
	$[2,3,4,4,2,3,1,4,5,4]^T$	5.87215963×10^1		
	$[3,2,3,4,2,3,1,5,5,4]^T$	5.85886706×10^1	5784	9685

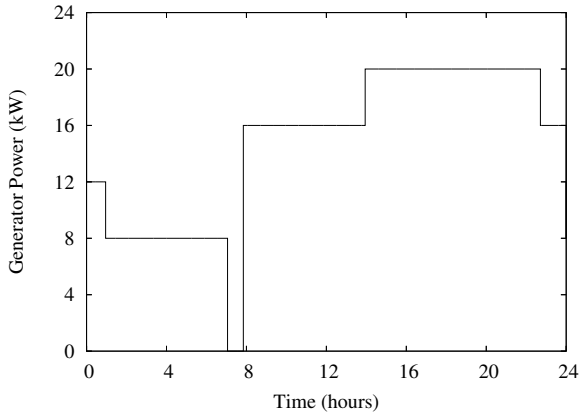

Figure 4.9: Optimal Generator Power Profile for 9 Switches.

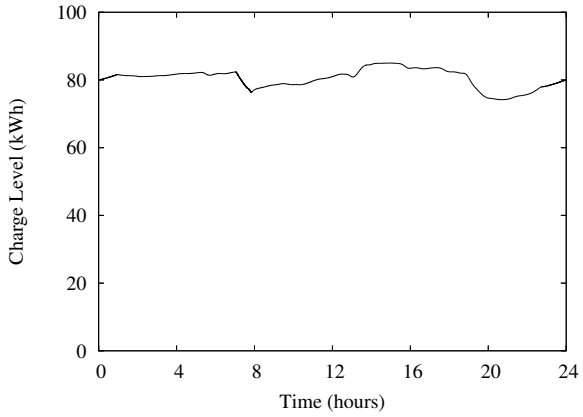

Figure 4.10: Optimal Battery Charge Profile for 9 Switches.

Table 4.4: Results for Solving the Model in [116].

Test	Minimum cost function value
1	8.87975339×10^1
2	6.01837893×10^1
3	6.51091281×10^1
4	6.06252775×10^1
5	6.01837944×10^1
6	6.03626756×10^1
7	6.03188550×10^1
8	6.33365059×10^1
9	6.01837982×10^1
10	8.87946446×10^1

The findings from Table 4.1-Table 4.3 reflect the findings of [150] that diesel generators are inefficient when they operate at a low load factor (around 40% - 50%) of their rated capacity. The findings also indicate that half of the operating time of the generator is spent on generating power during late afternoon and at night when the power source from the PV is not available. In addition, no significant difference is observed for the battery bank profiles among 4, 7, and 9 switches, where the charge level varies between 75 kWh and 85 kWh. A sharp fall of the battery charge level is also observed when the generator is turned off for a short period as demonstrated in Figure 4.10.

Table 4.4 shows the numerical results obtained by solving the transformed problem in [116] starting from ten random initial guesses. The findings in Table 4.4, when compared with our algorithm for 4 to 9 switches, show that our algorithm yields a better result compared with the approach in [116]. The best solution identified by our algorithm is 58.7216005 for 4 switches, compared with the best local minimum value of 60.1837893 identified from Table 4.4. Clearly, the method in [116] gets stuck in local minima and cannot determine a globally optimal solution. Note that we are using exactly the same objective function as the one used in [116], including the term penalizing short durations in a particular operating mode.

4.4 Alternative Hybrid Power System Simulation and Optimization Tools

HOMER [57] is a popular tool for preliminary design of hybrid power systems. It uses simple strategies with strong emphasis on economic factors to obtain an optimal design of a hybrid system by selecting the most appropriate system components. On the other hand, HYBRID2 [89] concentrates more on the technical characteristics of hybrid power systems and is able to optimize the operating strategies as well [57]. Barley et. al. [5] suggest to use HOMER in running a quick search to find the lowest life-cost of a hybrid power system from a range of possible operating strategies, whereas HYBRID2 is used to verify HOMER models for more accurate results. A third tool, implemented in the Matlab environment [39], was developed in [124]. This includes considerable details on various power flows, interaction of components, and applying the genetic algorithms to optimize the choice of system components as well as broad aspects of the operating strategies. Several actual systems were simulated and optimized to demonstrate the applicability of their tool [124]. However, no further development or application of the tool has appeared in the literature since [125].

A common feature of the above algorithms [57, 89, 124, 125] is that simulation is performed over relatively large time steps, typically at least 1 hour. This is done in order to simulate the system over at least several days to capture a variety of daily power demands and renewable power availability profiles. Smaller time steps would lead to excessively complex models under these circumstances. However, as can be seen from a typical load demand profile (see Figure 4.2), there is significant variation in the model inputs over a 1 hour period, and one would expect a similar level of variation for the optimal operating schedules within this period.

While only a crude cost function was proposed in [116], the operating cost for the model was shown to vary significantly with respect to the switching times for the diesel generator and other time dependent parameters [112]. Such sensitivities would not have been captured in the above models [57, 89, 124, 125] with

large time steps. In contrast to these models, [116] simulates a hybrid power system as a continuous time model that can capture the above mentioned variabilities. Fuel efficiency cost is represented by a nonlinear function in [116], while a linear relationship is used in both HOMER and HYBRID2. Next, both HOMER and HYBRID2 use the kinetic battery model to describe the charge and discharge rate, while [116] suggests a more basic formulation to represent these rates.

Calculating the total cost of operating the hybrid power system in HOMER and HYBRID2 is more comprehensive compared to [116], where only several surrogate terms were suggested. HOMER calculates the total net present cost (NPC) by incorporating the initial capital cost of the system components, replacement costs, maintenance costs, fuel costs, and costs of purchasing power from the grid. Likewise, HYBRID2 calculates the fixed and marginal costs of the system components as well as the economic parameters, such as interest and inflation rates.

The profiles of the load demand and renewable resource in [116] are based on data collected at quarterly intervals for 24 hours. An interpolation function is constructed to generate a continuous profile. For HOMER and HYBRID2, the load demand and renewable energy profiles are based on hourly data for up to a year.

4.5 Suggestions for a More Realistic Model

While our hybrid power system model is a specific example of a hybrid power system, the structure of the model is relatively simple and can be adapted to other system configurations.

The first limitation we discuss here is the modeling of the battery dynamics, in terms of recharge and discharge rates as represented by equations (4.1), (4.2) & (4.3), are not realistic in measuring the real cost of the battery. We intend to adopt the more realistic kinetic battery model of [88] to measure the recharge and discharge behavior. According to the kinetic battery model concept, a battery is modeled by a two-tank system: an available energy tank and a bound energy tank (see Figure 4.11). The available energy tank provides immediate energy for charg-

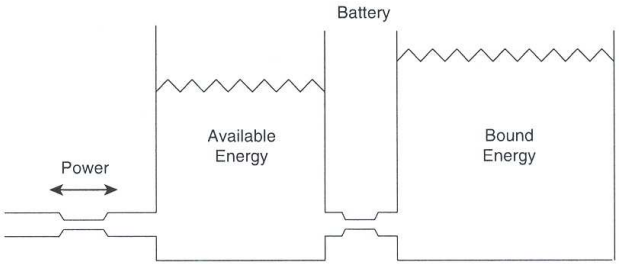

Figure 4.11: Kinetic Battery Model Concept.

ing or discharging, while the rest is chemically bound in the latter. The rate of conversion between two tanks depends on the difference in 'height' between these tanks. The mathematical formulations that describe the kinetic battery model are

$$\frac{dq_1}{dt} = -I - k'\Big(\frac{q_1}{c} - \frac{q_2}{1-c}\Big) \tag{4.19}$$

and

$$\frac{dq_2}{dt} = k'\Big(\frac{q_1}{c} - \frac{q_2}{1-c}\Big), \tag{4.20}$$

where $q_1 =$ available charge, $q_2 =$ bound charge, k' is a fixed conductance, c is the width of the available energy tank, and I is the current.

We intend to model the cost of battery usage more realistically by relating the daily use to the total lifetime. There are two common lifetime models for lead acid batteries: the post-processing models and the performance degradation models [7]. According to [7], the post-processing models are pure lifetime models used for assessing the impact of a particular operating scheme on the expected lifetime of the battery. Thus, these post-processing models can be used to analyze measured data from real systems. The performance degradation models combine either a charge transfer model or a voltage model with a typical lifetime model in such a way that the performance of the battery degrades as time goes by, depending on the utilization pattern of the battery. For the purpose of this paper, we discuss how to integrate the post-processing model into our optimal control problem only. We apply the Ah-throughput counting method to evaluate the lifetime consumption of the battery as the data of the total throughput is available and simple to apply to

approximate the cost of the battery.

Ah-throughput assumes that there is a fixed amount of energy that can be cycled through a battery before it requires replacement. The estimated throughput is derived from [7],

$$throughput = Average\{E_{nom} D_i C_{F,i}\}_Y^X, \tag{4.21}$$

where E_{nom} is the nominal battery capacity, D_i refers to the specific depth of discharge being considered, $C_{F,i}$ is the number of cycles to failure to the specific depth of discharge, i represents each depth of discharge measurement, and X to Y is the range over which the measurements of depth of discharge are taken. Note that the relationship between the depth of discharge and the number of cycles to the failure curve is provided by the manufacturer. Based on [7], the total throughput over a variety of discharge depth is approximately constant for most lead acid batteries. To adopt the Ah-throughput into the control optimal formulation, note that

$$x(t) = \int_0^T \frac{|\dot{C}(t)|}{2} dt \tag{4.22}$$

captures the total throughput of the battery bank over a daily time horizon. The cost of operating the battery bank over this time is then modeled by

$$C_{BB} = \frac{x(t)}{T_{TP}} C_B, \tag{4.23}$$

where T_{TP} is the total throughput over a battery bank lifetime and C_B is the cost of a battery bank.

The second limitation of the existing model is that forecasting of load demand and renewable power profiles is not carried out. It would be interesting to include the predictions of the future load demand and the forecasts of solar resource or other renewable resources as part of the control strategy of a hybrid power system. Some of the forecasting issues related to the solar/wind resources are size of the PV/wind

systems, daily temperature fluctuations, radiation forecasts, wind speed, humidity, ambient temperatures, observations of cloud cover and cloud movement, barometric pressure, and irradiation [151]. As for remote area electrification, size of the population, changes of consumer behavior, special community events, seasonal/short-term variation of environmental condition are among the factors which can bring significant changes to short-term and long-term load demand, as observed in [151]. Different load profiles, such as daily, weekly, or seasonal demand profiles on individual usage patterns should be considered when constructing a robust hybrid power system.

Thirdly, the existing model only focuses on the operating strategy of a discrete value diesel generator. Further study on a wide range of generators, such as variable speed generators or continuous type generators should be considered, where the output is not limited to discrete values only.

Fourthly, the power from renewable energy, i.e. PV arrays (2.5 kW), is considered small compared to the diesel generator (20 kW), where the latter is the backbone of the energy supply. A system that is based primarily on renewable resources, with the diesel generator as a backup supply, should be considered for long term usage due to increasing fuel costs and continually cheaper renewable supplies.

The existing formulation also neglects the initial setup cost of each component of the hybrid power system. It is vital to incorporate the initial capital cost of the system's components into the total cost of the hybrid power system to increase the efficiency of the system. This introduces discrete variables into the problem which complicate the optimization process considerably. Several algorithms in this regard have been proposed in the literature [125, 146].

4.6 Concluding Remarks

An optimal control problem for optimizing the operation of a hybrid power system is considered in this chapter. The problem is first formulated as a discrete-valued optimal control problem where the switching sequence as well as the switching times

for the discrete-valued control are to be determined. This problem is converted into a mixed discrete dynamic optimization problem by applying a modified time scaling transformation. It is then decomposed into a bi-level problem to facilitate the application of a discrete filled function method. A new metaheuristic approach which incorporates a discrete filled function algorithm into a standard optimal control software is proposed. The computational results have demonstrated that the method is capable of determining a significantly improved solution compared with the earlier approach in [116].

Chapter 5

Case Study: Sensor Scheduling System

We consider a general optimal sensor scheduling problem in this chapter, and propose a transformation to convert it into an equivalent mixed discrete optimization problem, as discussed in Section 5.3. Then, we adopt our proposed global optimization algorithm, which incorporates a discrete filled function method and a gradient-based method, to avoid local minima and speed up the computation. To evaluate the effectiveness of our algorithm, we solve a numerical example from the literature and compare the results with those obtained from the methods in [63] and [26] in Section 5.4.

5.1 Sensor Scheduling Problem

Sensors are used in various applications, including military surveillance, ground mapping, tracking and recognition of targets, instrumentation, air traffic control, imaging, and robotics [45]. Information collected by sensors is used to design activities that evolve over time in the underlying system [14]. For example, in a defense system, surveillance sensors are used to detect, identify, and localize targets, assess levels of threat, and deduce enemy intent [104]. In some applications, such

as robotics, operating several sensors simultaneously causes interference in the system and thus affects the measurement accuracy [16]. Consequently, it is impossible to operate all of the sensors at once. Instead, we need to schedule the operation of sensors over a given time frame so that the signal estimation error is minimized. We assume in this study that only one sensor may be active at any one time. The work presented here was motivated by [4] and [63]. In [4], the optimal scheduling policy is obtained by solving a quasi-variational inequality. However, the complexity of the model in [4] makes it difficult to compute an optimal solution. On the other hand, [63] considers open-loop policies with switches from one sensor to another. This reference proposes a time scaling transformation, which aims to capture a large variety of possible switching sequences. The sensor scheduling problem, which is formulated as a discrete-valued optimal control problem, is first transformed into an optimal parameter selection problem, and then solved using an existing optimal control software. The optimal control for the original problem is determined through a reverse transformation. However, this approach introduces a large number of artificial switches, many of which are not utilized in the optimal solution. As a consequence, the resulting optimization problem has many local minima. A study similar to that considered in [63] is performed in [26], where a combination of a branch and cut technique and a gradient-based method is applied to solve the continuous-time sensor scheduling problem.

We consider a general optimal sensor scheduling problem, which is similar to the one discussed in [63] and [26], and propose a transformation to convert it into an equivalent mixed discrete optimization problem. An algorithm similar to that in the previous chapter is then used to determine a near globally optimal solution.

5.2 Problem Formulation

Consider the following system of linear stochastic differential equations on a given probability space $(\Omega, \mathcal{F}, \mathcal{P})$:

$$d\mathbf{x}(t) = A(t)\mathbf{x}(t)dt + B(t)dK(t), \qquad t \in [0, T],$$

with initial condition

$$\mathbf{x}(0) = \mathbf{x}_0.$$

Here, $\{\mathbf{x}(t),\ t \in [0, T]\}$ is a $\mathbb{R}^n$-valued state process representing a signal of interest. It is assumed to be square integrable. The initial state, $\mathbf{x}_0$, is a $\mathbb{R}^n$-valued Gaussian random vector on $(\Omega, \mathcal{F}, \mathcal{P})$ with mean $\bar{\mathbf{x}}_0$ and covariance matrix P_0. Furthermore, $A : [0, T] \to \mathbb{R}^{n\times n}$ and $B : [0, T] \to \mathbb{R}^{n\times p}$ are continuous functions. The process $\{K(t),\ t \in [0, T]\}$ is a standard $\mathbb{R}^p$-valued Brownian motion on $(\Omega, \mathcal{F}, \mathcal{P})$ with mean zero and given covariance matrix $Q \in \mathbb{R}^{p\times p}$, where Q is symmetric and positive semi-definite.

Suppose that there are M sensors for detecting the state process. Only one of these sensors may be operated at any one time. A *sensor schedule* is a function $\phi : [0, T] \to \{1, \ldots, M\}$ that returns the active sensor at time t. In other words, $\phi(t) = i$ means sensor i is active at time t. Let Φ be the set of all measurable sensor schedules and let $\mathbf{y}$ be the observation process associated with the scheduling policy ϕ. For any $\phi \in \Phi$, we have the following output equation:

$$d\mathbf{y}(t) = \sum_{i=1}^{M} \chi_{\{t:\phi(t)=i\}}(t)\big\{C_i(t)x(t)dt + D_i(t)dW_i(t)\big\}, \qquad t \in [0, T],$$

and

$$\mathbf{y}(0) = \mathbf{0},$$

where, for each $\mathcal{I} \subset [0,T]$,

$$\chi_{\mathcal{I}}(t) = \begin{cases} 1, & t \in \mathcal{I}, \\ 0, & \text{otherwise}, \end{cases}$$

and $\{W_i(t),\ t \in [0,T]\}$ is a standard $\mathbb{R}^m$-valued Brownian motion with mean zero and covariance matrix $R \in \mathbb{R}^{m\times m}$, where R is symmetric and positive definite, $C_i : [0,T] \to \mathbb{R}^{m\times n}$ and $D_i : [0,T] \to \mathbb{R}^{m\times m}$ are continuous functions.

Each sensor makes an observation of the state process that is contaminated by noise. The history of such observation processes is denoted by $\{\mathbf{y}(s), 0 \le s \le t\}$. The data collected from the M sensors are used to estimate the state $\mathbf{x}$ at time t. The best estimate of $\mathbf{x}(t)$ is known as $\hat{\mathbf{x}}(t)$. Since $\mathbf{y}$ is corrupted by noise, the history observed is uncertain. Let the history of such a process be denoted by the smallest σ-algebra, $\mathcal{F}_t^{\mathbf{y}} = \sigma\{\mathbf{y}(s), 0 \le s \le t\}$. Hence, the optimal mean-square estimate of the state given $\mathcal{F}_t^{\mathbf{y}}$ is $\hat{\mathbf{x}}(t)$, and the associated error covariance is $P(t)$. Then, for a given $\phi \in \Phi$, the optimal $\hat{\mathbf{x}}(t)$ is given by the following theorem. The proof of this theorem may be found in [1].

Theorem 5.1 *For each sensor schedule $\phi \in \Phi$, the optimal mean-square estimate of the state $\hat{\mathbf{x}}(t)$ is the unique solution of the following stochastic differential equation:*

$$\begin{aligned} d\hat{\mathbf{x}}(t) &= \left[A(t) - P(t)\sum_{i=1}^{M} \chi_{\{t:\phi(t)=i\}}(t) C_i^{\top}(t)\bar{R}_i^{-1}(t)C_i(t)\right]\hat{\mathbf{x}}(t)dt \\ &\quad + \left[P(t)\sum_{i=1}^{M} \chi_{\{t:\phi(t)=i\}}(t) C_i^{\top}(t)\bar{R}_i^{-1}(t)\right] d\mathbf{y}(t), \quad t \in [0,T], \end{aligned} \tag{5.1}$$

and

$$\hat{\mathbf{x}}(0) = \bar{\mathbf{x}}_0, \tag{5.2}$$

where

$$\bar{R}_i^{-1}(t) = \left[D_i(t)R_i(t)D_i^{\top}(t)\right]^{-1}, \tag{5.3}$$

and the error covariance matrix $P : [0,T] \to \mathbb{R}^{n\times n}$ *is the unique solution of the matrix Riccati differential equation*

$$\begin{aligned}\dot{P}(t) &= A(t)P(t) + P(t)A^{\top}(t) + B(t)QB^{\top}(t) \\ &\quad - P(t)\sum_{i=1}^{M}\chi_{\{t:\phi(t)=i\}}(t)C_i^{\top}(t)\bar{R}_i^{-1}(t)C_i(t)P(t) \end{aligned} \tag{5.4}$$

with initial condition

$$P(0) = P_0. \tag{5.5}$$

Clearly, the solution of (5.4)-(5.5) depends on the sensor schedule that is chosen. Let $P(\cdot|\phi)$ be the solution corresponding to $\phi \in \Phi$. We formulate the following sensor scheduling problem.

Problem (P). Choose $\phi \in \Phi$ to minimize

$$g_0(\phi) = \alpha\text{trace}\{P(T|\phi)\} + \int_0^T \text{trace}\{P(t|\phi)\}dt, \tag{5.6}$$

subject to (5.4) and (5.5), where α is a non-negative constant.

The objective function (5.6) is designed to minimize the estimation error during the operation of the system. Note that Problem (P) is a discrete-valued optimal control problem. The main challenge in solving Problem (P) is that the control ϕ is constrained to take values in the discrete set $\{1,\ldots,M\}$. Each sensor schedule is completely determined by specifying the values in $\{1,\ldots,M\}$ that it assumes and the times when it switches from one value in $\{1,\ldots,M\}$ to another. Clearly, only a finite number of switches are able to be implemented in practice, and hence ϕ is a piecewise constant function with a finite number of switches. In other words, to solve Problem (P), we need to determine both the optimal switching sequence and the optimal switching times. Thus, we transform Problem (P) into an equivalent and solvable form in the next section.

5.3 Problem Transformation

Recall that only one sensor is active at each time and that only a finite number of switches are allowed. Suppose that we allow a sensor schedule ϕ to switch N times during the time horizon. Let $V = \{\mathbf{v} = [v_1, \ldots, v_{N+1}]^\top : \ v_i \in \{1, \ldots, M\}\}$ be the set of all possible switching sequence vectors. Let $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_{N+1}]^\top$, where $\sigma_i \geq 0, \ i = 1, \ldots, N+1$, denote the duration for which the corresponding sensor v_i in the sequence is active. Clearly,

$$\sum_{i=1}^{N+1} \sigma_i = T.$$

Let Σ denote the set of all such $\boldsymbol{\sigma}$. Note that under the assumption of a finite number of switches, N, any $\phi \in \Phi$ is completely determined by an element $(\mathbf{v}, \boldsymbol{\sigma}) \in V \times \Sigma$, where

$$\phi(t) = v_i, \qquad t \in \left[\sum_{j=1}^{i-1} \sigma_j, \ \sum_{j=1}^{i} \sigma_j \right], \qquad i = 1, \ldots, N+1.$$

We introduce a new time variable $\tau \in [0, N+1]$ and consider the fixed partition $\{0, 1, \ldots, N+1\}$. The original time horizon $[0, T]$ is transformed into the new time horizon $[0, N+1]$ as follows:

$$\dot{t}(\tau) = \sigma_i, \qquad \tau \in [i-1, i), \qquad i = 1, \ldots, N+1, \tag{5.7}$$

with the boundary conditions

$$t(0) = 0 \tag{5.8}$$

and

$$t(N+1) = T. \tag{5.9}$$

The original dynamics (5.4)-(5.5) are transformed into

$$\begin{aligned}\dot{\tilde{P}}(\tau) &= \sigma_i\Big[A(\tau)\tilde{P}(\tau)+\tilde{P}(\tau)A^\top(\tau)+B(\tau)QB^\top(\tau)\\ &\quad -\tilde{P}(\tau)C_{v_i}^\top(\tau)\bar{R}_{v_i}^{-1}(\tau)C_{v_i}(\tau)\tilde{P}(\tau)\Big],\\ &\quad \tau\in[i-1,i),\ i=1,\ldots,N+1,\end{aligned} \tag{5.10}$$

and

$$\tilde{P}(0)=P_0. \tag{5.11}$$

Hence, the transformed problem is stated formally below. Let $\tilde{P}(\cdot|\mathbf{v},\boldsymbol{\sigma})$ be the solution of (5.10)-(5.11) corresponding to $(\mathbf{v},\boldsymbol{\sigma})\in V\times\Sigma$.

Problem (R). Choose $\mathbf{v}\in V$ and $\boldsymbol{\sigma}\in\Sigma$ to minimize

$$g_0(\mathbf{v},\boldsymbol{\sigma})=\alpha\text{trace}\{\tilde{P}(N+1|\mathbf{v},\boldsymbol{\sigma})\}+\sum_{i=1}^{N+1}\int_{i-1}^{i}\text{trace}\{\tilde{P}(\tau|\mathbf{v},\boldsymbol{\sigma})\}\sigma_i\,d\tau, \tag{5.12}$$

subject to (5.7)-(5.9) and the dynamics (5.10)-(5.11), where α is a non-negative constant.

Problem (R), an equivalent problem to Problem (P), is a mixed discrete optimization problem with the discrete variable $\mathbf{v}$ representing the switching sequence and the continuous variable $\boldsymbol{\sigma}$ representing the time length of each mode. We propose to solve Problem (R) by first decomposing it into two levels. Note that for a fixed $\mathbf{v}\in V$, Problem (R) reduces to the following problem.

$\mathbf{Problem\ (R_1)}$. Given $\mathbf{v}\in V$, find a $\boldsymbol{\sigma}\in\Sigma$ to minimize

$$g_0(\boldsymbol{\sigma}|\mathbf{v})=\alpha\text{trace}\{\tilde{P}(N+1|\boldsymbol{\sigma},\mathbf{v})\}+\sum_{i=1}^{N+1}\int_{i-1}^{i}\text{trace}\{\tilde{P}(\tau|\boldsymbol{\sigma},\mathbf{v})\}\sigma_i\,d\tau, \tag{5.13}$$

subject to (5.7)-(5.9) and dynamics (5.10)-(5.11), where α is a non-negative con-

stant.

Problem (R_1) is a standard optimal parameter selection problem in a canonical form suitable for the application of a standard algorithm based on the control parameterization concept. For each given $\mathbf{v}$, the optimal value of g_0 in (5.13) can be determined using an optimal control software, such as MISER3.3, since the switching sequence is fixed. Note that in MISER3.3, the optimal parameter selection problem is solved using a sequential quadratic programming algorithm. The second problem in the proposed decomposition is defined as follows.

Problem $(\mathbf{R_2})$. Choose $\mathbf{v} \in V$ to minimize the objective function

$$J(\mathbf{v}), \tag{5.14}$$

where

$$J(\mathbf{v}) = \min_{\boldsymbol{\sigma} \in \Sigma} \quad g_0(\boldsymbol{\sigma}|\mathbf{v}).$$

Note that Problem (R_2) is a purely discrete optimization problem, but computing the value of $J(\mathbf{v})$ requires solving the corresponding Problem (R_1). Hence, Problem (R_1) is a subproblem of Problem (R_2). To obtain a near globally optimal solution for Problem (R), we propose a combined algorithm where Problem (R_2) will be solved using the discrete filled function method in Section 3.1 (i.e. Algorithm 3.2) and, at each iteration, Problem (R_1) is solved using MISER3.3. For our numerical computations, we have been able to incorporate the discrete filled function method within the MISER3.3 software. Note that we set $\rho_L = 0.001$ for our numerical computation to confirm a near global solution is attained from the algorithm.

Remark 5.1 *Note that the early time scale transformation proposed in* [63] *introduces a large number of artificial switching instants, typically $N \times M$, most of which are not used in the final optimal solution. As a result, the transformed problem yields many local minima, many of which have high objective values. Our*

method, similar to that in [25], avoids this difficulty because only N switches are needed.

5.4 Illustrative Example

Consider a sensor scheduling problem with six sensors and seven switches as discussed in [26]. Let $N = 7,\ M = 6,\ n = 2,\ m = 1,\ p = 2,\ T = 8,\ \alpha = 0,\ c = 0.5,\ \mu_0 = 0.1,\ \rho_0 = 0.1,\ \omega = 1,\ \rho_L = 0.001,\ \hat{\rho} = 0.1,\ \hat{\mu} = 0.1$ and consider the following dynamics:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_1(t) \end{bmatrix} = \begin{bmatrix} 0.5 & 1.0 \\ 1.0 & 0.5 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 2.0 \\ 2.0 \end{bmatrix} K(t),$$

$$\begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

where

$$P_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$C_1(t) = \begin{bmatrix} 1 + 1.2\sin(2t) & 0 \\ 1 + 1.2\sin(2t) & 0 \end{bmatrix}, \quad D_1(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_1(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$C_2(t) = \begin{bmatrix} 1 + 0.5\cos(2t) & 1 + 0.5\cos(2t) \\ 0 & 0 \end{bmatrix}, \quad D_2(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_2(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$C_3(t) = \begin{bmatrix} 1 + 0.5\sin(2t) & 0 \\ 0 & 1 + 0.5\cos(2t) \end{bmatrix}, \quad D_3(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_3(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$C_4(t) = \begin{bmatrix} 0 & 1 + 0.5\cos(2t) \\ 1 + 0.5\sin(2t) & 0 \end{bmatrix}, \quad D_4(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_4(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$C_5(t) = \begin{bmatrix} 0 & 0 \\ 1 + 0.5\cos(2t) & 1 + 0.5\sin(2t) \end{bmatrix}, \quad D_5(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_5(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$C_6(t) = \begin{bmatrix} 0 & 1 + 1.8\sin(2t) \\ 0 & 1 + 1.8\cos(2t) \end{bmatrix}, \quad D_6(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_6(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

For the ease of computation, we have been able to embed the filled function algorithm into the MISER3.3 program. The algorithm is terminated when $\mu = 1 \times 10^{-41}$ and $\rho = 1 \times 10^{-3}$, at which stage the best local minimizer found

cannot be improved. The computation is performed using the modified version of MISER3.3 on a Windows-based PC, with a CPU speed of 2.4GHz and 2GB RAM. We solve Problem (R), which has a total number of 1,679,616 potential switching sequences, using $\mathbf{v_0} = [6, 5, 2, 6, 5, 2, 6, 1]^\top$ as the initial sequence and $\boldsymbol{\sigma}_0 = [1, 1, 1, 1, 1, 1, 1, 1]^\top$ as the initial guess for $\boldsymbol{\sigma}$. Note that P_0 is initialized as a 2×2 identity matrix. Relevant results obtained are summarized in Table 5.1. The entries in the $\mathbf{v}^*$ column indicate the optimal solutions for the local searches. From Table 5.1, $\boldsymbol{\sigma}^* = [0.23501973, 0, 0, 7.7649803, 0, 0, 0, 0]^\top$ for the assumed global minimum indicates that sensors 2, 3, 4, and 5 are not used in the final optimal solution during the tenth iteration. Hence, only two out of six sensors are turned on. The assumed global optimal switching sequence is to turn on sensor 1, followed by sensor 6, with the objective function 14.33176. The number of original function evaluations and filled function evaluations are 5293 and 8517, respectively. This represents 0.32% of the total number of potential sequences. Note that the objective function evaluations do not include those that were obtained from the look-up table.

We tested the problem with five different initial sequences. These are $[1, 2, 3, 4, 5, 6, 1, 2]^\top$, $[6, 5, 4, 3, 2, 1, 6, 5]^\top$, $[1, 6, 3, 2, 4, 5, 3, 1]^\top$, $[1, 6, 1, 6, 1, 6, 1, 6]^\top$, and $[6, 6, 1, 2, 5, 4, 2, 1]^\top$, using the same P_0 and $\boldsymbol{\sigma}_0$ as in the first computation. As many as fifty local minima are found during the searches from the various initial sequences. Starting at each initial sequence, the algorithm successfully identified the same assumed discrete global minimum sequence of Problem (R) observed in the first experiment, that is, sensor 1 is followed by sensor 6, with the cost function value $J = 14.33176$. Again, computational results show that only up to 0.32% of the total number of potential sequences are evaluated. The optimal operating schedule for the control and states are depicted in Figure 5.1. In addition, several different choices of P_0 are tested in our experimentation with various initial switching sequences. The optimal operating scheme for $P_0 = \mathbf{0},\ P_0 = 6I,\ P_0 = 10I$ are illustrated by Figures 5.2, 5.3, and 5.4, respectively. From these graphs, only the first and sixth sensors are ever used, while the other four are not utilized in any optimal solution.

Table 5.1: Numerical Results for $P_0 = I$.

$\mathbf{v}^*$	$\boldsymbol{\sigma}^*$	J
$[1,6,1,1,6,1,6,6]^\top$	$[0.24035917,0,0,0,0,0,7.7525870,0.0070538593]^\top$	14.649680367412879
$[6,1,6,6,1,6,1,1]^\top$	$[0.17566501,0.18470974,0,7.6396253,0,0,0,0]^\top$	14.504334985710470
$[1,6,1,6,6,1,1,6]^\top$	$[0.23511799,0,0,7.7648820,0,0,0,0]^\top$	14.331763146735220
$[1,6,2,6,6,2,2,6]^\top$	$[0.23501894,0,0,7.7649811,0,0,0,0]^\top$	14.331763102479558
$[1,6,6,6,6,3,3,5]^\top$	$[0.23502083,0,0,7.7649792,0,0,0,0]^\top$	14.331763102474610
$[1,6,6,6,6,6,5,5]^\top$	$[0.23502039,0,0,7.7649796,0,0,0,0]^\top$	14.331763102473506
$[1,6,6,6,1,5,6,2]^\top$	$[0.23501994,0,0,7.7649801,0,0,0,0]^\top$	14.331763102471598
$[1,1,6,6,6,6,5,1]^\top$	$[0.23501894,0,0,7.7649811,0,0,0,0]^\top$	14.331763102445281
$[1,1,6,6,5,6,6,2]^\top$	$[0.23501979,0,0,7.7649802,0,0,0,0]^\top$	14.331763102440952
$[1,1,6,6,6,5,2,1]^\top$	$[0.23501973,0,0,7.7649803,0,0,0,0]^\top$	14.331763102437696

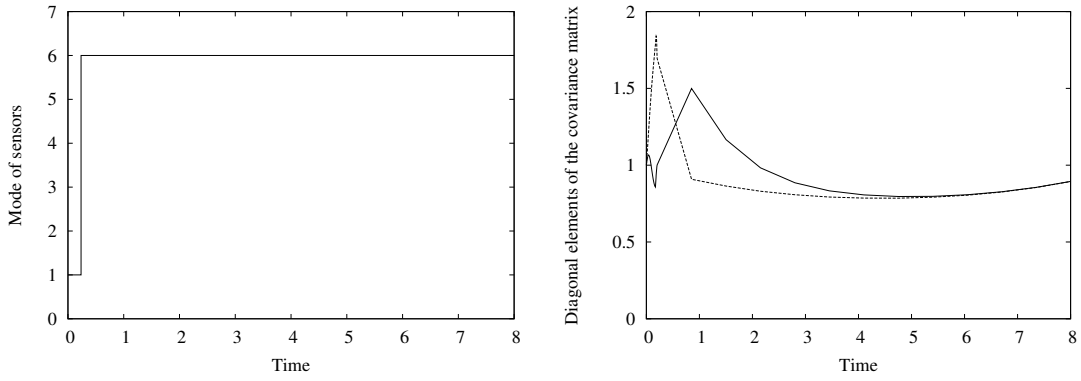

Figure 5.1: Optimal Sensor Operating Scheme with $P_0 = I$.

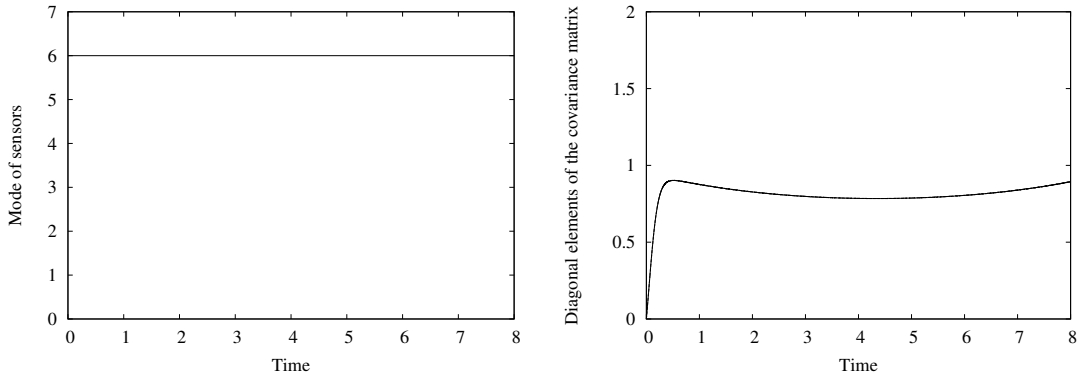

Figure 5.2: Optimal Sensor Operating Scheme with $P_0 = \mathbf{0}$.

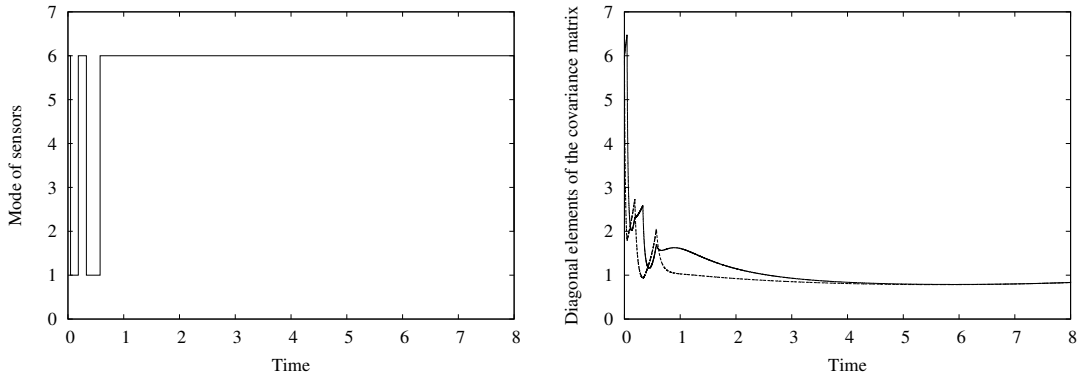

Figure 5.3: Optimal Sensor Operating Scheme with $P_0 = 6I$.

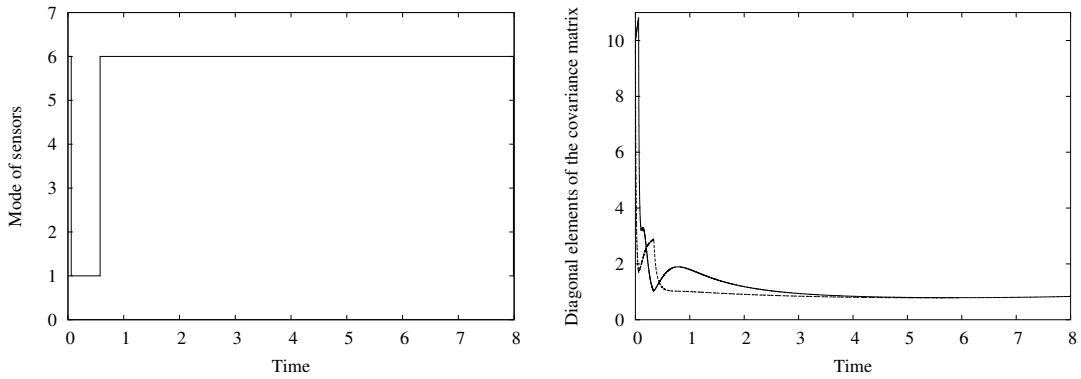

Figure 5.4: Optimal Sensor Operating Scheme with $P_0 = 10I$.

Table 5.2: A Comparison of Numerical Results with Other Methods.

Methods	Objective values
Method in [26] with $P_0 = \mathbf{0}$	19.6553
Method in [63] with $P_0 = 10I$	19.2353622
Proposed method with $P_0 = 10I$	16.5697177
Proposed method with $P_0 = 6I$	15.8781106
Proposed method with $P_0 = I$	14.3317631
Proposed method with $P_0 = \mathbf{0}$	12.9949699

We also compare the solutions obtained here with those obtained from the other methods proposed in [63] and [26]. These results are summarized in Table 5.2. Note that the error estimation found with the proposed algorithm is significantly lower than 19.6553, the optimal solution reported in [26], which was obtained using a combination of a branch and bound technique with a gradient-based method. To the best of our knowledge, $P_0 = \mathbf{0}$ is used in [26]. Note that any non-zero choices of P_0 lead to even higher objective values when used in conjunction with the solution in [26].

5.5 Concluding Remarks

A sensor scheduling problem is considered in this chapter. It is formulated as a discrete-valued optimal control problem and then transformed into a mixed discrete optimization problem. Then, it is decomposed into a bi-level problem. A new metaheuristic approach, similar to that in Chapter 4, which incorporates the discrete filled function algorithm into a standard optimal control software, is proposed for finding a global solution of this problem. Numerical results show that the method is efficient, reliable, and robust in solving a complex discrete-valued optimal control problem. The proposed method successfully identified significantly improved solutions compared with other methods available in the literature. Note that, unlike the hybrid power system problem in Chapter 4, this application problem does not lead to any infeasible subproblems, so there is no need to assign artificially high cost values to a sequence resulting in an infeasible subproblem.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This study has demonstrated and established the effectiveness of a computational procedure for determining near global solutions for some classes of discrete and mixed discrete optimization problems. Most of the practical discrete and mixed discrete optimization problems are nonlinear and known to have more than one locally optimal solution. This suggests the need for global optimization techniques which seek the best solution amongst multiple local optima. In this thesis, our attention is focused on developing a metaheuristic technique to determine the global optimal solution of discrete-valued optimal control problems. Our metaheuristic approach is based on the combination of a discrete filled function method and a computational optimal control algorithm.

Various discrete filled function methods are reviewed in this thesis. The fundamental idea behind the filled function concept is to introduce an auxiliary function to move from a current local minimizer to an improved point, if it exists. Interestingly, each filled function has its own termination and parameter updating criteria. We have been able to give a generic algorithm in this thesis which allows us to capture their commonalities and also to contrast their differences. Based on the theoretical properties of the various methods and our own computational implementations,

we found that the discrete filled function method in [106] seems to perform best for bound constrained problems.

Further, we propose several variations of the method in [106] to try and enhance its computational efficiency. Two of these algorithms, namely Algorithms 3.4 and 3.5, appear to be significantly more efficient than the standard method in terms of the number of objective function evaluations. However, both methods frequently fail to identify the global solution due to the choice of a random set of points used to initialize the search of a local minimum of the filled function. In other words, the gain in efficiency for these two algorithms is offset by reduced reliability. After analyzing the numerical results, we choose to adopt the standard algorithm from [106] in combination with a computational optimal algorithm to solve a general class of discrete-valued optimal control problems.

To determine an optimal discrete-valued control, we need to determine the order of the switching sequence and the times at which these switches take place. However, conventional computational optimal control approaches are designed for solving problems in which the control takes values in a convex set, and thus these approaches cannot solve a discrete-valued optimal control problem directly. To overcome these difficulties, we propose a new transformation to convert a discrete-valued optimal control problem into an equivalent mixed discrete optimization problem. This transformation introduces a new discrete variable to represent the switching sequences and a new continuous variable to represent the switching times. To facilitate with the application of our proposed global optimization algorithm, we decompose the mixed problem into a bi-level problem. The upper level problem in this decomposition is a purely discrete optimization problem, where the discrete global switching sequence is determined using a discrete filled function approach. The subproblem is simply a standard optimal control problem where the objective function value is determined using MISER3.3, an optimal control software based on the concept of control parameterization. To increase the computational efficiency, we construct a look-up table to store each value of the objective function of the subproblem computed so far, thus avoiding the need to repeatedly solve the sub-

problem at the same point. This is essential to the computational efficiency because the numerical solution of the subproblem requires considerable computational time.

We apply the proposed global algorithm to solve a hybrid power system control problem and a sensor scheduling problem. Both of these are fairly complex practical application problems which have been demonstrated to possess many locally optimal solutions. For the numerical implementation, we have incorporated the discrete filled function technique directly into the MISER3.3 software. Numerical results suggest that the method is efficient, reliable, and robust in solving both complex discrete-valued optimal control problems. In fact, the proposed method successfully identified significantly improved solutions compared with those obtained from other methods available in the literature.

6.2 Limitations of the Study

Note that the discrete filled function algorithm we adopted from [106] is designed for a box constrained (or linear inequality constrained) problem where the feasible search region is pathwise connected and has easily identifiable vertices. These properties are not necessarily met in the application of the algorithm to solve the hybrid power system problem because the feasibility of a point is not known until an attempt has been made to solve the corresponding subproblem. Although we do not remove such a point from the search region directly, we assign artificially high cost to it. It may well be the case that the effective feasible region of subproblem becomes non-convex and non-connected. However, it is difficult to ascertain this behavior beforehand and our application of the algorithm to the subproblem must hence be viewed as a metaheuristic approach. On the other hand, we are essentially employing a penalty method to deal with complex constraints which would be difficult to incorporate directly at the upper level. This simple idea seems to work well in practice and warrants some further investigation in the future.

In addition, a sequential quadratic programming method is employed within MISER3.3 to solve the subproblems of both discrete-valued optimization problems.

This is a local search method and thus cannot guarantee the global optimality for the solution of the subproblem. In other words, although we aim to solve the upper level problem globally, the lower level problem may only yield a locally optimal solution. Therefore, we again consider our approach to be a metaheuristic global optimization method with no implied guarantee of finding the overall global optimum. Nevertheless, numerical results demonstrate that good quality solutions can be determined effectively compared with other methods in the literature.

6.3 Future Work

Discrete filled function methods form an active area of research open to further investigation and improvements in solving mixed discrete optimization problems globally. It would be interesting to test different procedures for minimizing the filled function, starting at points other than the immediate neighbourhood of the current local minimizer, as often suggested in the literature.

Secondly, it would of benefit to implement a global optimization method to solve the subproblem resulting from the original discrete-valued optimal control problem globally [69]. Continuous filled function methods are available, but these are mainly aimed at solving unconstrained problems. Constraints at the subproblem level would hence require the use of penalty methods.

Thirdly, refinements of the hybrid power system model so that it reflects an actual operating environment more realistically are required in the future. These include a more realistic battery model, forecasting tool for load demand and renewable power profiles, considering a wide range of generators (both variable speed generators or continuous type) and allowing a broader class of controls (eg. smooth rather then piecewise constant).

Fourthly, a comparison between the discrete filled function method and other metaheuristic approaches, such as greedy search, simulated annealing, and genetic algorithm, would be an interesting future research direction in combinatorial optimization. This is likely to involve extensive computational studies, though, since

each class of algorithm allows significant variations and tuning of algorithm parameters.

In addition, it is certainly possible and may well be worthwhile to modify MISER3.3 so that the embedded discrete filled function algorithm can be readily invoked by non-expert users when solving any discrete valued optimal control problem. Besides, other variations of discrete filled function methods, such as those proposed in Sections 3.2-3.6, could also be implemented in application problems to see if improved results can be attained. Lastly, the application of the proposed method to other practical discrete-valued optimal control problems would be interesting, particularly the well studied submarine transit path problem [11].

Appendix A

FORTRAN Codes for the Algorithms in Chapter 3

A.1 Algorithm 3.2

```
      MODULE constants
      SAVE
      INTEGER,PARAMETER::n=5,m=2*n,totalpoints=161051
      DOUBLEPRECISION,PARAMETER::c=0.5d0,tao=1.0d0,rhol=0.001,muhat=0.1d0,rhohat=0.1d0
      INTEGER::xl(n),xu(n),e(n,m),countf,countff,countxstar,pointsCalculated
      DOUBLEPRECISION::mu,rho,table(totalpoints,n+1)
      END MODULE constants

      PROGRAM standard
      USE constants
      IMPLICIT NONE
      EXTERNAL::minf,minp
      INTEGER::x(n),x0(n),xstar(n),newx(n),neighbour(n),i,j,k,flag
      DOUBLEPRECISION::f,fxbest


! Set the initial of the parameters
      mu=0.1d0
      rho=0.1d0

! Initialize the lookup table counter.
      pointsCalculated = 0

! Define the upper and lower bounds for x(n)
      xl=-5
      xu=5

! Set the initial count for f, G, local minimizer obtained.
      countf=0
      countff=0
      countxstar=0

! Set the search direction
```

```
      DO i=1,n
         DO j=1,m
            IF (i.EQ.j) THEN
               e(i,j)=1
            ELSEIF (j.EQ.i+n) THEN
               e(i,j)=-1
            ELSE
               e(i,j)=0
            ENDIF
         ENDDO
      ENDDO

! Set the initial value of x(n)
1000  IF (countxstar.EQ.0) THEN
          x0(:)= 5
      ELSE
          x0(:)= newx(:)
      ENDIF
      PRINT*,"initial point,x0=",x0

! Call the local search of the original function
      CALL minf(x0,xstar,fxbest)

! Display the minimal solution & value of the original function
      PRINT*,"x*=",xstar
      PRINT*,"f(x*)=",fxbest

! Call the local search of the filled function
      CALL minp(xstar,newx,flag)

! Display the output
      PRINT*,"the number of function evaluations=",countf
      PRINT*,"the number of filled function evaluations=",countff
      PRINT*,"mu=",mu,"rho",rho

      IF (flag.EQ.1) THEN
         PRINT*,"Point in a lower basin is found as f(x)<f(x*)"
         PRINT*,"new starting point,x=",newx,",f(x)=",f(newx)
         countxstar=countxstar+1
         GOTO 1000
      ELSE
         PRINT*,"x*=",xstar,"is the global solution."
      ENDIF

      END


! Define the LOGICAL FUNCTION feasible
      LOGICAL FUNCTION feasible(point)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::point(n)
      INTEGER::i

      feasible=.TRUE.

      DO i=1,n
         IF (point(i).GT.xu(i).OR.point(i).LT.xl(i)) THEN
             feasible=.FALSE.
             RETURN
         ENDIF
      ENDDO

      END FUNCTION feasible


! Objective function
```

```
      DOUBLEPRECISION FUNCTION f(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,j
      LOGICAL:: indicator

      DO i=1,pointsCalculated
          indicator = .TRUE.
          DO j=1,n
              IF (table(i,j) .NE. x(j)) THEN
                  indicator = .FALSE.
                  EXIT
              ENDIF
          ENDDO

          IF (indicator) THEN
              f = table(i,n+1)
              RETURN
          ENDIF
      ENDDO

      f=0.0d0
      DO i=1,n-1
         f=f+(100.0d0*(x(i+1)-x(i)**2)**2+(1.0d0-x(i))**2)
      ENDDO

      table(pointsCalculated+1,1:n) = x(:)
      table(pointsCalculated+1,n+1) = f

      pointsCalculated = pointsCalculated + 1
      countf=countf+1

      END FUNCTION f


! Filled function
      DOUBLEPRECISION FUNCTION p(x,xstar,fx,fxstar)
      USE constants
      IMPLICIT NONE
      INTEGER::i,s
      INTEGER,INTENT(IN)::x(n),xstar(n)
      DOUBLEPRECISION,INTENT(IN)::fx,fxstar
      DOUBLEPRECISION::f,y,v,a

      y=fx-fxstar

      s=0
      DO i=1,n
         s=s+(x(i)-xstar(i))**2
      ENDDO

      v=mu*((1.0d0-c)*((1.0d0-c*mu)/(mu-c*mu))**(-y/tao)+c)

      a=y*v
      p=a*y-rho*s
      countff=countff+1

      END FUNCTION p


! Define the LOGICAL FUNCTION to check if a vertex exists
      LOGICAL FUNCTION vertex(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,tally
```

```
      tally=0

      DO i=1,n
         IF (x(i).EQ.xl(i).OR.x(i).EQ.xu(i)) THEN
             tally=tally+1
         ENDIF
      ENDDO

      IF (tally.EQ.n) THEN
          vertex=.TRUE.
      ELSE
          vertex=.FALSE.
      ENDIF

      END FUNCTION vertex


! Local search of the original function,f
      SUBROUTINE minf(x0,xstar,fxbest)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x0(n)
      INTEGER,INTENT(OUT)::xstar(n)
      DOUBLEPRECISION,INTENT(OUT)::fxbest
      INTEGER::x(n),xbest(n),neighbour(n),i,j
      DOUBLEPRECISION::f,fx,temp
      LOGICAL::feasible

      x(:)=x0(:)

      DO

        fx=f(x)
        xbest(:)=x(:)
        fxbest=fx

        DO j=1,2*n
           neighbour(:)=x(:)+e(:,j)

           IF (feasible(neighbour)) THEN
               temp=f(neighbour)
               IF (temp.LT.fxbest) THEN
                   xbest(:)=neighbour(:)
                   fxbest=temp
               ENDIF
           ENDIF
        ENDDO

        IF (fxbest.EQ.fx) THEN
            xstar(:)=x(:)
            RETURN
        ELSE
            x(:)=xbest(:)
        ENDIF

      ENDDO

      END SUBROUTINE minf


! Local search of the filled function.
      SUBROUTINE minp(xstar,newx,flag)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN):: xstar(n)
      INTEGER,INTENT(OUT):: newx(n),flag
```

```
INTEGER:: x(n),XI(n,m),j
DOUBLEPRECISION:: fxstar,f
LOGICAL::feasible
EXTERNAL::minp1

DO j=1,m
      XI(:,j) = xstar(:) + e(:,j)
ENDDO

fxstar = f(xstar)

DO

  IF (rho .LT. rhol) THEN
      flag = 2
      RETURN
  ENDIF

DO j=1,m
   x(:) = XI(:,j)
   IF (feasible(x)) THEN
       CALL minp1(xstar,fxstar,x,flag)
       IF (flag .EQ. 1) THEN
           newx(:) = x(:)
           RETURN
       ENDIF
   ENDIF
ENDDO

rho = rho*rhohat

ENDDO

END SUBROUTINE minp


SUBROUTINE minp1(xstar,fxstar,x,flag)
USE constants
IMPLICIT NONE
INTEGER,INTENT(IN):: xstar(n)
DOUBLEPRECISION,INTENT(IN):: fxstar
INTEGER,INTENT(INOUT):: x(n)
INTEGER,INTENT(OUT):: flag
INTEGER::xbest(n),neighbour(n),betterxbest(n),j,flag1
DOUBLEPRECISION:: f,fx,fcurrent,fn,p,pcurrent,pbest,temp,total,totalbest
LOGICAL::feasible,vertex

flag = 0

DO

   xbest(:) = x(:)
   fcurrent = f(x)
   pcurrent = p(x,xstar,fcurrent,fxstar)
   pbest = pcurrent

   flag1 = 0

   DO j=1,2*n
       neighbour(:) = x(:) + e(:,j)

       IF (feasible(neighbour)) THEN
           fn = f(neighbour)
           IF (fn .LT. fxstar) THEN
               flag = 1
               x(:) = neighbour(:)
               RETURN
```

```
                ELSE
                    temp = p(neighbour,xstar,fn,fxstar)
                    IF (temp .LT. pbest) THEN
                         xbest(:) = neighbour(:)
                         pbest = temp
                    ENDIF

                    IF (temp .LT. pcurrent .AND. fn .LT. fcurrent) THEN
                        IF (flag1 .eq. 0) THEN
                            betterxbest(:) = neighbour(:)
                            totalbest = temp + fn
                            flag1 = 1
                        ELSE
                            total = temp + fn
                            IF (total .LT. totalbest) THEN
                                totalbest = total
                                betterxbest(:) = neighbour(:)
                            ENDIF
                        ENDIF
                    ENDIF
                ENDIF
            ENDIF

        ENDDO

        IF (pbest .EQ. pcurrent) THEN
            IF (vertex(x)) THEN
                RETURN
            ELSE
                mu = mu*muhat
                CYCLE
            ENDIF
        ELSE
            IF (flag1 .EQ. 1) THEN
                x(:) = betterxbest(:)
            ELSE
                x(:) = xbest(:)
            ENDIF
        ENDIF

    ENDDO

    END SUBROUTINE minp1
```

A.2 Algorithm 3.3

```
    MODULE constants
    SAVE
    INTEGER,PARAMETER::n=5,m=2*n,totalpoints=161051
    DOUBLEPRECISION,PARAMETER::c=0.5d0,tao=1.0d0,rhol=0.001,muhat=0.1d0,rhohat=0.1d0
    INTEGER::xl(n),xu(n),e(n,m),countf,countff,countxstar,pointsCalculated
    DOUBLEPRECISION::mu,rho,table(totalpoints,n+1)
    END MODULE constants

    PROGRAM variation1
    USE constants
    IMPLICIT NONE
    EXTERNAL::minf,minp
    INTEGER::x(n),x0(n),xstar(n),newx(n),neighbour(n),i,j,k,flag
    DOUBLEPRECISION::f,fxbest


! Set the initial of the parameters
```

```
      mu=0.1d0
      rho=0.1d0

! Initialize the lookup table counter.
      pointsCalculated = 0

! Define the upper and lower bounds for x(n)
      xl=-5
      xu=5

! Set the initial count for f, G, local minimizer obtained.
      countf=0
      countff=0
      countxstar=0

! Set the search direction
      DO i=1,n
         DO j=1,m
            IF (i.EQ.j) THEN
               e(i,j)=1
            ELSEIF (j.EQ.i+n) THEN
               e(i,j)=-1
            ELSE
               e(i,j)=0
            ENDIF
         ENDDO
      ENDDO

! Set the initial value of x(n)
1000  IF (countxstar.EQ.0) THEN
          x0(:)= 5
      ELSE
          x0(:)= newx(:)
      ENDIF
      PRINT*,"initial point,x0=",x0

! Call the local search of the original function
      CALL minf(x0,xstar,fxbest)

! Display the minimal solution & value of the original function
      PRINT*,"x*=",xstar
      PRINT*,"f(x*)=",fxbest

! Call the local search of the filled function
      CALL minp(xstar,newx,flag)

! Display the output
      PRINT*,"the number of function evaluations=",countf
      PRINT*,"the number of filled function evaluations=",countff
      PRINT*,"mu=",mu,"rho",rho

      IF (flag.EQ.1) THEN
         PRINT*,"Point in a lower basin is found as f(x)<f(x*)"
         PRINT*,"new starting point,x=",newx,",f(x)=",f(newx)
         countxstar=countxstar+1
         GOTO 1000
      ELSE
         PRINT*,"x*=",xstar,"is the global solution."
      ENDIF

      END


! Define the LOGICAL FUNCTION feasible
      LOGICAL FUNCTION feasible(point)
      USE constants
      IMPLICIT NONE
```

```
      INTEGER,INTENT(IN)::point(n)
      INTEGER::i

      feasible=.TRUE.

      DO i=1,n
         IF (point(i).GT.xu(i).OR.point(i).LT.xl(i)) THEN
             feasible=.FALSE.
             RETURN
         ENDIF
      ENDDO

      END FUNCTION feasible


! Objective function
      DOUBLEPRECISION FUNCTION f(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,j
      LOGICAL:: indicator

      DO i=1,pointsCalculated
          indicator = .TRUE.
          DO j=1,n
              IF (table(i,j) .NE. x(j)) THEN
                  indicator = .FALSE.
                  EXIT
              ENDIF
          ENDDO

          IF (indicator) THEN
              f = table(i,n+1)
              RETURN
          ENDIF
      ENDDO

      f=0.0d0
      DO i=1,n-1
         f=f+(100.0d0*(x(i+1)-x(i)**2)**2+(1.0d0-x(i))**2)
      ENDDO

      table(pointsCalculated+1,1:n) = x(:)
      table(pointsCalculated+1,n+1) = f

      pointsCalculated = pointsCalculated + 1
      countf=countf+1

      END FUNCTION f


! Filled function
      DOUBLEPRECISION FUNCTION p(x,xstar,fx,fxstar)
      USE constants
      IMPLICIT NONE
      INTEGER::i,s
      INTEGER,INTENT(IN)::x(n),xstar(n)
      DOUBLEPRECISION,INTENT(IN)::fx,fxstar
      DOUBLEPRECISION::f,y,v,a

      y=fx-fxstar

      s=0
      DO i=1,n
         s=s+(x(i)-xstar(i))**2
      ENDDO
```

```
      v=mu*((1.0d0-c)*((1.0d0-c*mu)/(mu-c*mu))**(-y/tao)+c)

      a=y*v
      p=a*y-rho*s
      countff=countff+1

      END FUNCTION p


! Define the LOGICAL FUNCTION to check if a vertex exists
      LOGICAL FUNCTION vertex(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,tally

      tally=0

      DO i=1,n
         IF (x(i).EQ.xl(i).OR.x(i).EQ.xu(i)) THEN
             tally=tally+1
         ENDIF
      ENDDO

      IF (tally.EQ.n) THEN
          vertex=.TRUE.
      ELSE
          vertex=.FALSE.
      ENDIF

      END FUNCTION vertex


! Local search of the original function,f
      SUBROUTINE minf(x0,xstar,fxbest)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x0(n)
      INTEGER,INTENT(OUT)::xstar(n)
      DOUBLEPRECISION,INTENT(OUT)::fxbest
      INTEGER::x(n),xbest(n),neighbour(n),i,j
      DOUBLEPRECISION::f,fx,temp
      LOGICAL::feasible

      x(:)=x0(:)

      DO

        fx=f(x)
        xbest(:)=x(:)
        fxbest=fx

        DO j=1,2*n
           neighbour(:)=x(:)+e(:,j)

           IF (feasible(neighbour)) THEN
               temp=f(neighbour)
               IF (temp.LT.fxbest) THEN
                   xbest(:)=neighbour(:)
                   fxbest=temp
               ENDIF
           ENDIF
        ENDDO

        IF (fxbest.EQ.fx) THEN
            xstar(:)=x(:)
```

```
            RETURN
          ELSE
              x(:)=xbest(:)
          ENDIF

        ENDDO

        END SUBROUTINE minf


  ! Local search of the filled function.
        SUBROUTINE minp(xstar,newx,flag)
        USE constants
        IMPLICIT NONE
        INTEGER,INTENT(IN):: xstar(n)
        INTEGER,INTENT(OUT):: newx(n),flag
        INTEGER:: x(n),XI(n,m),j
        DOUBLEPRECISION:: fxstar,f,fx
        LOGICAL::feasible
        EXTERNAL::minp1

        fxstar = f(xstar)

        DO j=1,m
           XI(:,j) = xstar(:) + 2*e(:,j)
           x(:) = XI(:,j)
           IF (feasible(x)) THEN
               fx=f(x)
               IF (fx .LT. fxstar) THEN
                   flag = 1
                   newx(:) = x(:)
                   RETURN
               ENDIF
           ENDIF
        ENDDO

        DO

           IF (rho .LT. rhol) THEN
               flag = 2
               RETURN
           ENDIF

           DO j=1,m
               x(:) = XI(:,j)
               IF (feasible(x)) THEN
                   CALL minp1(xstar,fxstar,x,flag)
                   IF (flag .EQ. 1) THEN
                       newx(:) = x(:)
                       RETURN
                   ENDIF
               ENDIF
           ENDDO

           rho = rho*rhohat

        ENDDO

        END SUBROUTINE minp


        SUBROUTINE minp1(xstar,fxstar,x,flag)
        USE constants
        IMPLICIT NONE
        INTEGER,INTENT(IN):: xstar(n)
        DOUBLEPRECISION,INTENT(IN):: fxstar
        INTEGER,INTENT(INOUT):: x(n)
```

```
INTEGER,INTENT(OUT):: flag
INTEGER::xbest(n),neighbour(n),betterxbest(n),j,flag1
DOUBLEPRECISION:: f,fx,fcurrent,fn,p,pcurrent,pbest,temp,total,totalbest
LOGICAL::feasible,vertex

flag = 0

DO

   xbest(:) = x(:)
   fcurrent = f(x)
   pcurrent = p(x,xstar,fcurrent,fxstar)
   pbest = pcurrent

   flag1 = 0

   DO j=1,2*n
       neighbour(:) = x(:) + e(:,j)

       IF (feasible(neighbour)) THEN
           fn = f(neighbour)
           IF (fn .LT. fxstar) THEN
               flag = 1
               x(:) = neighbour(:)
               RETURN
           ELSE
               temp = p(neighbour,xstar,fn,fxstar)
               IF (temp .LT. pbest) THEN
                    xbest(:) = neighbour(:)
                    pbest = temp
               ENDIF

               IF (temp .LT. pcurrent .AND. fn .LT. fcurrent) THEN
                   IF (flag1 .eq. 0) THEN
                       betterxbest(:) = neighbour(:)
                       totalbest = temp + fn
                       flag1 = 1
                   ELSE
                       total = temp + fn
                       IF (total .LT. totalbest) THEN
                           totalbest = total
                           betterxbest(:) = neighbour(:)
                       ENDIF
                   ENDIF
               ENDIF
           ENDIF
       ENDIF

   ENDDO

   IF (pbest .EQ. pcurrent) THEN
       IF (vertex(x)) THEN
           RETURN
       ELSE
           mu = mu*muhat
           CYCLE
       ENDIF
   ELSE
       IF (flag1 .EQ. 1) THEN
           x(:) = betterxbest(:)
       ELSE
           x(:) = xbest(:)
       ENDIF
   ENDIF

ENDDO
```

```
      END SUBROUTINE minp1
```

A.3 Algorithm 3.4

```
      MODULE constants
      SAVE
      INTEGER,PARAMETER::n=5,m=2*n,totalpoints=161051
      DOUBLEPRECISION,PARAMETER::c=0.5d0,tao=1.0d0,rhol=0.001,muhat=0.1d0,rhohat=0.1d0
      INTEGER::xl(n),xu(n),e(n,m),countf,countff,countxstar,pointsCalculated
      DOUBLEPRECISION::mu,rho,table(totalpoints,n+1)
      END MODULE constants

      PROGRAM variation2
      USE constants
      IMPLICIT NONE
      EXTERNAL::minf,minp
      INTEGER::x(n),x0(n),xstar(n),newx(n),neighbour(n),i,j,k,flag
      DOUBLEPRECISION::f,fxbest


! Set the initial of the parameters
      mu=0.1d0
      rho=0.1d0

! Initialize the lookup table counter.
      pointsCalculated = 0

! Define the upper and lower bounds for x(n)
      xl=-5
      xu=5

! Set the initial count for f, G, local minimizer obtained.
      countf=0
      countff=0
      countxstar=0

! Set the search direction
      DO i=1,n
         DO j=1,m
            IF (i.EQ.j) THEN
               e(i,j)=1
            ELSEIF (j.EQ.i+n) THEN
               e(i,j)=-1
            ELSE
               e(i,j)=0
            ENDIF
         ENDDO
      ENDDO

! Set the initial value of x(n)
1000  IF (countxstar.EQ.0) THEN
          x0(:)= 5
      ELSE
          x0(:)= newx(:)
      ENDIF
      PRINT*,"initial point,x0=",x0

! Call the local search of the original function
      CALL minf(x0,xstar,fxbest)

! Display the minimal solution & value of the original function
      PRINT*,"x*=",xstar
```

```
      PRINT*,"f(x*)=",fxbest

! Call the local search of the filled function
      CALL minp(xstar,newx,flag)

! Display the output
      PRINT*,"the number of function evaluations=",countf
      PRINT*,"the number of filled function evaluations=",countff
      PRINT*,"mu=",mu,"rho",rho

      IF (flag.EQ.1) THEN
         PRINT*,"Point in a lower basin is found as f(x)<f(x*)"
         PRINT*,"new starting point,x=",newx,",f(x)=",f(newx)
         countxstar=countxstar+1
         GOTO 1000
      ELSE
         PRINT*,"x*=",xstar,"is the global solution."
      ENDIF

      END


! Define the LOGICAL FUNCTION feasible
      LOGICAL FUNCTION feasible(point)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::point(n)
      INTEGER::i

      feasible=.TRUE.

      DO i=1,n
         IF (point(i).GT.xu(i).OR.point(i).LT.xl(i)) THEN
             feasible=.FALSE.
             RETURN
         ENDIF
      ENDDO

      END FUNCTION feasible


! Objective function
      DOUBLEPRECISION FUNCTION f(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,j
      LOGICAL:: indicator

      DO i=1,pointsCalculated
          indicator = .TRUE.
          DO j=1,n
              IF (table(i,j) .NE. x(j)) THEN
                  indicator = .FALSE.
                  EXIT
              ENDIF
          ENDDO

          IF (indicator) THEN
              f = table(i,n+1)
              RETURN
          ENDIF
      ENDDO

      f=0.0d0
      DO i=1,n-1
         f=f+(100.0d0*(x(i+1)-x(i)**2)**2+(1.0d0-x(i))**2)
```

```
      ENDDO

      table(pointsCalculated+1,1:n) = x(:)
      table(pointsCalculated+1,n+1) = f

      pointsCalculated = pointsCalculated + 1
      countf=countf+1

      END FUNCTION f


! Filled function
      DOUBLEPRECISION FUNCTION p(x,xstar,fx,fxstar)
      USE constants
      IMPLICIT NONE
      INTEGER::i,s
      INTEGER,INTENT(IN)::x(n),xstar(n)
      DOUBLEPRECISION,INTENT(IN)::fx,fxstar
      DOUBLEPRECISION::f,y,v,a

      y=fx-fxstar

      s=0
      DO i=1,n
         s=s+(x(i)-xstar(i))**2
      ENDDO

      v=mu*((1.0d0-c)*((1.0d0-c*mu)/(mu-c*mu))**(-y/tao)+c)

      a=y*v
      p=a*y-rho*s
      countff=countff+1

      END FUNCTION p


! Define the LOGICAL FUNCTION to check if a vertex exists
      LOGICAL FUNCTION vertex(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,tally

      tally=0

      DO i=1,n
         IF (x(i).EQ.xl(i).OR.x(i).EQ.xu(i)) THEN
             tally=tally+1
         ENDIF
      ENDDO

      IF (tally.EQ.n) THEN
          vertex=.TRUE.
      ELSE
          vertex=.FALSE.
      ENDIF

      END FUNCTION vertex


! Local search of the original function,f
      SUBROUTINE minf(x0,xstar,fxbest)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x0(n)
      INTEGER,INTENT(OUT)::xstar(n)
      DOUBLEPRECISION,INTENT(OUT)::fxbest
```

```
      INTEGER::x(n),xbest(n),neighbour(n),i,j
      DOUBLEPRECISION::f,fx,temp
      LOGICAL::feasible

      x(:)=x0(:)

      DO

        fx=f(x)
        xbest(:)=x(:)
        fxbest=fx

        DO j=1,2*n
           neighbour(:)=x(:)+e(:,j)

           IF (feasible(neighbour)) THEN
               temp=f(neighbour)
               IF (temp.LT.fxbest) THEN
                   xbest(:)=neighbour(:)
                   fxbest=temp
               ENDIF
           ENDIF
        ENDDO

        IF (fxbest.EQ.fx) THEN
            xstar(:)=x(:)
            RETURN
        ELSE
            x(:)=xbest(:)
        ENDIF

      ENDDO

      END SUBROUTINE minf


! Local search of the filled function.
      SUBROUTINE minp(xstar,newx,flag)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN):: xstar(n)
      INTEGER,INTENT(OUT):: newx(n),flag
      INTEGER:: x(n),XI(n,m),i,j,FLOOR,timeArray(3),xbest(n)
      DOUBLEPRECISION:: fxstar,f,fx,RAND,fxbest
      LOGICAL:: feasible
      EXTERNAL::minp1

      fxbest = 1.0d5
      fxstar = f(xstar)

! Generate a set of random points from FORTRAN.
      CALL itime(timeArray)
      DO j=1,m
         DO i=1,n
            XI(i,j)=FLOOR(xl(i)+(RAND(timeArray(3)+j+i))*(xu(i)-xl(i)+1))
            x(:) = XI(:,j)
         ENDDO
         PRINT*,"random point,x=", XI(:,j),"f(x)=",f(x)
      ENDDO

      fxstar = f(xstar)

      DO

         IF (rho .LT. rhol) THEN
             flag = 2
             RETURN
```

```
   ENDIF

   DO j=1,m
      x(:) = XI(:,j)
      fx=f(x)
      IF (fx .LT. fxstar) THEN
          flag = 1
          newx(:) = x(:)
          RETURN
       ENDIF
   ENDDO

   DO j=1,m
      x(:) = XI(:,j)
      CALL minp1(xstar,fxstar,x,flag)
      IF (flag .EQ. 1) THEN
          newx(:) = x(:)
          RETURN
      ENDIF
   ENDDO

   rho = rho*rhohat

ENDDO

END SUBROUTINE minp


SUBROUTINE minp1(xstar,fxstar,x,flag)
USE constants
IMPLICIT NONE
INTEGER,INTENT(IN):: xstar(n)
DOUBLEPRECISION,INTENT(IN):: fxstar
INTEGER,INTENT(INOUT):: x(n)
INTEGER,INTENT(OUT):: flag
INTEGER::xbest(n),neighbour(n),betterxbest(n),j,flag1
DOUBLEPRECISION:: f,fx,fcurrent,fn,p,pcurrent,pbest,temp,total,totalbest
LOGICAL::feasible,vertex

flag = 0

DO

   xbest(:) = x(:)
   fcurrent = f(x)
   pcurrent = p(x,xstar,fcurrent,fxstar)
   pbest = pcurrent

   flag1 = 0

   DO j=1,2*n
       neighbour(:) = x(:) + e(:,j)

       IF (feasible(neighbour)) THEN
           fn = f(neighbour)
           IF (fn .LT. fxstar) THEN
               flag = 1
               x(:) = neighbour(:)
               RETURN
           ELSE
               temp = p(neighbour,xstar,fn,fxstar)
               IF (temp .LT. pbest) THEN
                    xbest(:) = neighbour(:)
                    pbest = temp
               ENDIF

               IF (temp .LT. pcurrent .AND. fn .LT. fcurrent) THEN
```

```
                    IF (flag1 .eq. 0) THEN
                        betterxbest(:) = neighbour(:)
                        totalbest = temp + fn
                        flag1 = 1
                    ELSE
                        total = temp + fn
                        IF (total .LT. totalbest) THEN
                            totalbest = total
                            betterxbest(:) = neighbour(:)
                        ENDIF
                    ENDIF
                ENDIF
            ENDIF
        ENDIF

    ENDDO

    IF (pbest .EQ. pcurrent) THEN
        IF (vertex(x)) THEN
            RETURN
        ELSE
            mu = mu*muhat
            CYCLE
        ENDIF
    ELSE
        IF (flag1 .EQ. 1) THEN
            x(:) = betterxbest(:)
        ELSE
            x(:) = xbest(:)
        ENDIF
    ENDIF

ENDDO

END SUBROUTINE minp1
```

A.4 Algorithm 3.5

```
MODULE constants
SAVE
INTEGER,PARAMETER::n=5,m=2*n,totalpoints=161051
DOUBLEPRECISION,PARAMETER::c=0.5d0,tao=1.0d0,rhol=0.001,muhat=0.1d0,rhohat=0.1d0
INTEGER::xl(n),xu(n),e(n,m),countf,countff,countxstar,pointsCalculated
DOUBLEPRECISION::mu,rho,table(totalpoints,n+1)
END MODULE constants

PROGRAM variation3
USE constants
IMPLICIT NONE
EXTERNAL::minf,minp
INTEGER::x(n),x0(n),xstar(n),newx(n),neighbour(n),i,j,k,flag
DOUBLEPRECISION::f,fxbest


! Set the initial of the parameters
     mu=0.1d0
     rho=0.1d0

! Initialize the lookup table counter.
     pointsCalculated = 0

! Define the upper and lower bounds for x(n)
     xl=-5
```

```
      xu=5

! Set the initial count for f, G, local minimizer obtained.
      countf=0
      countff=0
      countxstar=0

! Set the search direction
      DO i=1,n
         DO j=1,m
            IF (i.EQ.j) THEN
               e(i,j)=1
            ELSEIF (j.EQ.i+n) THEN
               e(i,j)=-1
            ELSE
               e(i,j)=0
            ENDIF
         ENDDO
      ENDDO

! Set the initial value of x(n)
1000  IF (countxstar.EQ.0) THEN
          x0(:)= 5
      ELSE
          x0(:)= newx(:)
      ENDIF
      PRINT*,"initial point,x0=",x0

! Call the local search of the original function
      CALL minf(x0,xstar,fxbest)

! Display the minimal solution & value of the original function
      PRINT*,"x*=",xstar
      PRINT*,"f(x*)=",fxbest

! Call the local search of the filled function
      CALL minp(xstar,newx,flag)

! Display the output
      PRINT*,"the number of function evaluations=",countf
      PRINT*,"the number of filled function evaluations=",countff
      PRINT*,"mu=",mu,"rho",rho

      IF (flag.EQ.1) THEN
         PRINT*,"Point in a lower basin is found as f(x)<f(x*)"
         PRINT*,"new starting point,x=",newx,",f(x)=",f(newx)
         countxstar=countxstar+1
         GOTO 1000
      ELSE
         PRINT*,"x*=",xstar,"is the global solution."
      ENDIF

      END


! Define the LOGICAL FUNCTION feasible
      LOGICAL FUNCTION feasible(point)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::point(n)
      INTEGER::i

      feasible=.TRUE.

      DO i=1,n
         IF (point(i).GT.xu(i).OR.point(i).LT.xl(i)) THEN
             feasible=.FALSE.
```

```
            RETURN
         ENDIF
      ENDDO

      END FUNCTION feasible


! Objective function
      DOUBLEPRECISION FUNCTION f(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,j
      LOGICAL:: indicator

      DO i=1,pointsCalculated
          indicator = .TRUE.
          DO j=1,n
              IF (table(i,j) .NE. x(j)) THEN
                  indicator = .FALSE.
                  EXIT
              ENDIF
          ENDDO

          IF (indicator) THEN
              f = table(i,n+1)
              RETURN
          ENDIF
      ENDDO

      f=0.0d0
      DO i=1,n-1
         f=f+(100.0d0*(x(i+1)-x(i)**2)**2+(1.0d0-x(i))**2)
      ENDDO

      table(pointsCalculated+1,1:n) = x(:)
      table(pointsCalculated+1,n+1) = f

      pointsCalculated = pointsCalculated + 1
      countf=countf+1

      END FUNCTION f


! Filled function
      DOUBLEPRECISION FUNCTION p(x,xstar,fx,fxstar)
      USE constants
      IMPLICIT NONE
      INTEGER::i,s
      INTEGER,INTENT(IN)::x(n),xstar(n)
      DOUBLEPRECISION,INTENT(IN)::fx,fxstar
      DOUBLEPRECISION::f,y,v,a

      y=fx-fxstar

      s=0
      DO i=1,n
         s=s+(x(i)-xstar(i))**2
      ENDDO

      v=mu*((1.0d0-c)*((1.0d0-c*mu)/(mu-c*mu))**(-y/tao)+c)

      a=y*v
      p=a*y-rho*s
      countff=countff+1

      END FUNCTION p
```

```
! Define the LOGICAL FUNCTION to check if a vertex exists
      LOGICAL FUNCTION vertex(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,tally

      tally=0

      DO i=1,n
         IF (x(i).EQ.xl(i).OR.x(i).EQ.xu(i)) THEN
             tally=tally+1
         ENDIF
      ENDDO

      IF (tally.EQ.n) THEN
          vertex=.TRUE.
      ELSE
          vertex=.FALSE.
      ENDIF

      END FUNCTION vertex


! Local search of the original function,f
      SUBROUTINE minf(x0,xstar,fxbest)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x0(n)
      INTEGER,INTENT(OUT)::xstar(n)
      DOUBLEPRECISION,INTENT(OUT)::fxbest
      INTEGER::x(n),xbest(n),neighbour(n),i,j
      DOUBLEPRECISION::f,fx,temp
      LOGICAL::feasible

      x(:)=x0(:)

      DO

        fx=f(x)
        xbest(:)=x(:)
        fxbest=fx

        DO j=1,2*n
           neighbour(:)=x(:)+e(:,j)

           IF (feasible(neighbour)) THEN
               temp=f(neighbour)
               IF (temp.LT.fxbest) THEN
                   xbest(:)=neighbour(:)
                   fxbest=temp
               ENDIF
           ENDIF
        ENDDO

        IF (fxbest.EQ.fx) THEN
            xstar(:)=x(:)
            RETURN
        ELSE
            x(:)=xbest(:)
        ENDIF

      ENDDO

      END SUBROUTINE minf
```

```
! Local search of the filled function.
      SUBROUTINE minp(xstar,newx,flag)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN):: xstar(n)
      INTEGER,INTENT(OUT):: newx(n),flag
      INTEGER:: x(n),XI(n,m),i,j,FLOOR,timeArray(3),xbest(n)
      DOUBLEPRECISION:: fxstar,f,fx,RAND,fxbest
      LOGICAL:: feasible
      EXTERNAL::minp1

      fxbest = 1.0d5
      fxstar = f(xstar)

! Generate a set of random points from FORTRAN.
      CALL itime(timeArray)
      DO j=1,m
         DO i=1,n
            XI(i,j)=FLOOR(xl(i)+(RAND(timeArray(3)+j+i))*(xu(i)-xl(i)+1))
            x(:) = XI(:,j)
            IF (feasible(x)) THEN
                fx=f(x)
                IF (fx .LT. fxbest) THEN
                    xbest(:)=x(:)
                    fxbest=fx
                ENDIF
            ENDIF
         ENDDO
          PRINT*,"random point,x=", XI(:,j),"f(x)=",f(x)
      ENDDO


      IF (fxbest .LT. fxstar) THEN
          flag = 1
          newx(:) = xbest(:)
          RETURN
      ENDIF

      DO

         IF (rho .LT. rhol) THEN
             flag = 2
             RETURN
         ENDIF

         DO j=1,m
            x(:) = XI(:,j)
            CALL minp1(xstar,fxstar,x,flag)
            IF (flag .EQ. 1) THEN
                newx(:) = x(:)
                RETURN
            ENDIF
         ENDDO

         rho = rho*rhohat

      ENDDO

      END SUBROUTINE minp


      SUBROUTINE minp1(xstar,fxstar,x,flag)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN):: xstar(n)
```

```
DOUBLEPRECISION,INTENT(IN):: fxstar
INTEGER,INTENT(INOUT):: x(n)
INTEGER,INTENT(OUT):: flag
INTEGER::xbest(n),neighbour(n),betterxbest(n),j,flag1
DOUBLEPRECISION:: f,fx,fcurrent,fn,p,pcurrent,pbest,temp,total,totalbest
LOGICAL::feasible,vertex

flag = 0

DO

   xbest(:) = x(:)
   fcurrent = f(x)
   pcurrent = p(x,xstar,fcurrent,fxstar)
   pbest = pcurrent

   flag1 = 0

   DO j=1,2*n
       neighbour(:) = x(:) + e(:,j)

       IF (feasible(neighbour)) THEN
           fn = f(neighbour)
           IF (fn .LT. fxstar) THEN
               flag = 1
               x(:) = neighbour(:)
               RETURN
           ELSE
               temp = p(neighbour,xstar,fn,fxstar)
               IF (temp .LT. pbest) THEN
                    xbest(:) = neighbour(:)
                    pbest = temp
               ENDIF

               IF (temp .LT. pcurrent .AND. fn .LT. fcurrent) THEN
                   IF (flag1 .eq. 0) THEN
                       betterxbest(:) = neighbour(:)
                       totalbest = temp + fn
                       flag1 = 1
                   ELSE
                       total = temp + fn
                       IF (total .LT. totalbest) THEN
                           totalbest = total
                           betterxbest(:) = neighbour(:)
                       ENDIF
                   ENDIF
               ENDIF
           ENDIF
       ENDIF

   ENDDO

   IF (pbest .EQ. pcurrent) THEN
       IF (vertex(x)) THEN
           RETURN
       ELSE
           mu = mu*muhat
           CYCLE
       ENDIF
   ELSE
       IF (flag1 .EQ. 1) THEN
           x(:) = betterxbest(:)
       ELSE
           x(:) = xbest(:)
       ENDIF
   ENDIF
```

```
      ENDDO

      END SUBROUTINE minp1
```

A.5 Algorithm 3.6

```
      MODULE constants
      SAVE
      INTEGER,PARAMETER::n=5,m=2*n,totalpoints=161051
      DOUBLEPRECISION,PARAMETER::c=0.5d0,tao=1.0d0,rhol=0.001,muhat=0.1d0,rhohat=0.1d0
      INTEGER::xl(n),xu(n),e(n,m),countf,countff,countxstar,pointsCalculated
      DOUBLEPRECISION::mu,rho,table(totalpoints,n+1)
      END MODULE constants

      PROGRAM variation4
      USE constants
      IMPLICIT NONE
      EXTERNAL::minf,minp
      INTEGER::x(n),x0(n),xstar(n),newx(n),neighbour(n),i,j,k,flag
      DOUBLEPRECISION::f,fxbest


! Set the initial of the parameters
      mu=0.1d0
      rho=0.1d0

! Initialize the lookup table counter.
      pointsCalculated = 0

! Define the upper and lower bounds for x(n)
      xl=-5
      xu=5

! Set the initial count for f, G, local minimizer obtained.
      countf=0
      countff=0
      countxstar=0

! Set the search direction
      DO i=1,n
         DO j=1,m
            IF (i.EQ.j) THEN
               e(i,j)=1
            ELSEIF (j.EQ.i+n) THEN
               e(i,j)=-1
            ELSE
               e(i,j)=0
            ENDIF
         ENDDO
      ENDDO

! Set the initial value of x(n)
1000  IF (countxstar.EQ.0) THEN
          x0(:)= 5
      ELSE
          x0(:)= newx(:)
      ENDIF
      PRINT*,"initial point,x0=",x0

! Call the local search of the original function
      CALL minf(x0,xstar,fxbest)

! Display the minimal solution & value of the original function
```

```
      PRINT*,"x*=",xstar
      PRINT*,"f(x*)=",fxbest

! Call the local search of the filled function
      CALL minp(xstar,newx,flag)

! Display the output
      PRINT*,"the number of function evaluations=",countf
      PRINT*,"the number of filled function evaluations=",countff
      PRINT*,"mu=",mu,"rho",rho

      IF (flag.EQ.1) THEN
         PRINT*,"Point in a lower basin is found as f(x)<f(x*)"
         PRINT*,"new starting point,x=",newx,",f(x)=",f(newx)
         countxstar=countxstar+1
         GOTO 1000
      ELSE
         PRINT*,"x*=",xstar,"is the global solution."
      ENDIF

      END


! Define the LOGICAL FUNCTION feasible
      LOGICAL FUNCTION feasible(point)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::point(n)
      INTEGER::i

      feasible=.TRUE.

      DO i=1,n
         IF (point(i).GT.xu(i).OR.point(i).LT.xl(i)) THEN
             feasible=.FALSE.
             RETURN
         ENDIF
      ENDDO

      END FUNCTION feasible


! Objective function
      DOUBLEPRECISION FUNCTION f(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,j
      LOGICAL:: indicator

      DO i=1,pointsCalculated
          indicator = .TRUE.
          DO j=1,n
              IF (table(i,j) .NE. x(j)) THEN
                  indicator = .FALSE.
                  EXIT
              ENDIF
          ENDDO

          IF (indicator) THEN
              f = table(i,n+1)
              RETURN
          ENDIF
      ENDDO

      f=0.0d0
      DO i=1,n-1
```

```
         f=f+(100.0d0*(x(i+1)-x(i)**2)**2+(1.0d0-x(i))**2)
      ENDDO

      table(pointsCalculated+1,1:n) = x(:)
      table(pointsCalculated+1,n+1) = f

      pointsCalculated = pointsCalculated + 1
      countf=countf+1

      END FUNCTION f


! Filled function
      DOUBLEPRECISION FUNCTION p(x,xstar,fx,fxstar)
      USE constants
      IMPLICIT NONE
      INTEGER::i,s
      INTEGER,INTENT(IN)::x(n),xstar(n)
      DOUBLEPRECISION,INTENT(IN)::fx,fxstar
      DOUBLEPRECISION::f,y,v,a

      y=fx-fxstar

      s=0
      DO i=1,n
         s=s+(x(i)-xstar(i))**2
      ENDDO

      v=mu*((1.0d0-c)*((1.0d0-c*mu)/(mu-c*mu))**(-y/tao)+c)

      a=y*v
      p=a*y-rho*s
      countff=countff+1

      END FUNCTION p


! Define the LOGICAL FUNCTION to check if a vertex exists
      LOGICAL FUNCTION vertex(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,tally

      tally=0

      DO i=1,n
         IF (x(i).EQ.xl(i).OR.x(i).EQ.xu(i)) THEN
             tally=tally+1
         ENDIF
      ENDDO

      IF (tally.EQ.n) THEN
          vertex=.TRUE.
      ELSE
          vertex=.FALSE.
      ENDIF

      END FUNCTION vertex


! Local search of the original function,f
      SUBROUTINE minf(x0,xstar,fxbest)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x0(n)
      INTEGER,INTENT(OUT)::xstar(n)
```

```
      DOUBLEPRECISION,INTENT(OUT)::fxbest
      INTEGER::x(n),xbest(n),neighbour(n),i,j
      DOUBLEPRECISION::f,fx,temp
      LOGICAL::feasible

      x(:)=x0(:)

      DO

        fx=f(x)
        xbest(:)=x(:)
        fxbest=fx

        DO j=1,2*n
           neighbour(:)=x(:)+e(:,j)

           IF (feasible(neighbour)) THEN
               temp=f(neighbour)
               IF (temp.LT.fxbest) THEN
                   xbest(:)=neighbour(:)
                   fxbest=temp
               ENDIF
           ENDIF
        ENDDO

        IF (fxbest.EQ.fx) THEN
            xstar(:)=x(:)
            RETURN
        ELSE
            x(:)=xbest(:)
        ENDIF

      ENDDO

      END SUBROUTINE minf


! Local search of the filled function.
      SUBROUTINE minp(xstar,newx,flag)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN):: xstar(n)
      INTEGER,INTENT(OUT):: newx(n),flag
      INTEGER:: x(n),i,j,FLOOR,timeArray(3),r(n,m),xr(n),XI(n,m)
      DOUBLEPRECISION:: fxstar,f,fx,RAND
      LOGICAL:: feasible
      EXTERNAL::minp1

! Generate a set of random points from FORTRAN.
      CALL itime(timeArray)
      DO j=1,m
         DO i=1,n
            r(i,j)=FLOOR(xl(i)+(RAND(timeArray(3)+j+i))*(xu(i)-xl(i)+1))
         ENDDO
      ENDDO

      fxstar = f(xstar)

      DO j=1,m
         IF (feasible(r(:,j))) THEN
             xr(:)=r(:,j)
             fx=f(xr(:))
             PRINT*,"feasible random point=",xr(:),"and f=",fx
             IF (fx .LT. fxstar) THEN
                 flag = 1
                 newx(:) = xr(:)
                 RETURN
```

```
       ENDIF
   ENDIF
ENDDO

DO
   IF (rho .LT. rhol) THEN
       flag = 2
       RETURN
   ENDIF

DO j=1,m
   XI(:,j) = xstar(:) + 2*e(:,j)
      x(:) = XI(:,j)
      IF (feasible(x)) THEN
          fx=f(x)
          IF (fx .LT. fxstar) THEN
              flag = 1
              newx(:) = x(:)
              RETURN
          ENDIF
      ENDIF
ENDDO

DO j=1,m
   x(:) = xstar(:) + 2*e(:,j)
   IF (feasible(x)) THEN
       CALL minp1(xstar,fxstar,x,flag)
       IF (flag .EQ. 1) THEN
           newx(:) = x(:)
           RETURN
       ENDIF
   ENDIF
ENDDO

rho = rho*rhohat

ENDDO

END SUBROUTINE minp


SUBROUTINE minp1(xstar,fxstar,x,flag)
USE constants
IMPLICIT NONE
INTEGER,INTENT(IN):: xstar(n)
DOUBLEPRECISION,INTENT(IN):: fxstar
INTEGER,INTENT(INOUT):: x(n)
INTEGER,INTENT(OUT):: flag
INTEGER::xbest(n),neighbour(n),betterxbest(n),j,flag1
DOUBLEPRECISION:: f,fx,fcurrent,fn,p,pcurrent,pbest,temp,total,totalbest
LOGICAL::feasible,vertex

flag = 0

DO

   xbest(:) = x(:)
   fcurrent = f(x)
   pcurrent = p(x,xstar,fcurrent,fxstar)
   pbest = pcurrent

   flag1 = 0

   DO j=1,2*n
       neighbour(:) = x(:) + e(:,j)

       IF (feasible(neighbour)) THEN
```

```
            fn = f(neighbour)
            IF (fn .LT. fxstar) THEN
                flag = 1
                x(:) = neighbour(:)
                RETURN
            ELSE
                temp = p(neighbour,xstar,fn,fxstar)
                IF (temp .LT. pbest) THEN
                     xbest(:) = neighbour(:)
                     pbest = temp
                ENDIF

                IF (temp .LT. pcurrent .AND. fn .LT. fcurrent) THEN
                    IF (flag1 .eq. 0) THEN
                        betterxbest(:) = neighbour(:)
                        totalbest = temp + fn
                        flag1 = 1
                    ELSE
                        total = temp + fn
                        IF (total .LT. totalbest) THEN
                            totalbest = total
                            betterxbest(:) = neighbour(:)
                        ENDIF
                    ENDIF
                ENDIF
            ENDIF
        ENDIF

    ENDDO

    IF (pbest .EQ. pcurrent) THEN
        IF (vertex(x)) THEN
            RETURN
        ELSE
            mu = mu*muhat
            CYCLE
        ENDIF
    ELSE
        IF (flag1 .EQ. 1) THEN
            x(:) = betterxbest(:)
        ELSE
            x(:) = xbest(:)
        ENDIF
    ENDIF

ENDDO

END SUBROUTINE minp1
```

A.6 Algorithm 3.7

```
MODULE constants
SAVE
INTEGER,PARAMETER::n=5,m=2*n,totalpoints=161051
DOUBLEPRECISION,PARAMETER::c=0.5d0,tao=1.0d0,rhol=0.001,muhat=0.1d0,rhohat=0.1d0
INTEGER::xl(n),xu(n),e(n,m),countf,countff,countxstar,pointsCalculated
DOUBLEPRECISION::mu,rho,table(totalpoints,n+1)
END MODULE constants

PROGRAM variation5
USE constants
IMPLICIT NONE
EXTERNAL::minf,minp
```

```
      INTEGER::x(n),x0(n),xstar(n),newx(n),neighbour(n),i,j,k,flag
      DOUBLEPRECISION::f,fxbest


! Set the initial of the parameters
      mu=0.1d0
      rho=0.1d0

! Initialize the lookup table counter.
      pointsCalculated = 0

! Define the upper and lower bounds for x(n)
      xl=-5
      xu=5

! Set the initial count for f, G, local minimizer obtained.
      countf=0
      countff=0
      countxstar=0

! Set the search direction
      DO i=1,n
         DO j=1,m
            IF (i.EQ.j) THEN
               e(i,j)=1
            ELSEIF (j.EQ.i+n) THEN
               e(i,j)=-1
            ELSE
               e(i,j)=0
            ENDIF
         ENDDO
      ENDDO

! Set the initial value of x(n)
1000  IF (countxstar.EQ.0) THEN
          x0(:)= 5
      ELSE
          x0(:)= newx(:)
      ENDIF
      PRINT*,"initial point,x0=",x0

! Call the local search of the original function
      CALL minf(x0,xstar,fxbest)

! Display the minimal solution & value of the original function
      PRINT*,"x*=",xstar
      PRINT*,"f(x*)=",fxbest

! Call the local search of the filled function
      CALL minp(xstar,newx,flag)

! Display the output
      PRINT*,"the number of function evaluations=",countf
      PRINT*,"the number of filled function evaluations=",countff
      PRINT*,"mu=",mu,"rho",rho

      IF (flag.EQ.1) THEN
         PRINT*,"Point in a lower basin is found as f(x)<f(x*)"
         PRINT*,"new starting point,x=",newx,",f(x)=",f(newx)
         countxstar=countxstar+1
         GOTO 1000
      ELSE
         PRINT*,"x*=",xstar,"is the global solution."
      ENDIF

      END
```

```
! Define the LOGICAL FUNCTION feasible
      LOGICAL FUNCTION feasible(point)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::point(n)
      INTEGER::i

      feasible=.TRUE.

      DO i=1,n
         IF (point(i).GT.xu(i).OR.point(i).LT.xl(i)) THEN
             feasible=.FALSE.
             RETURN
         ENDIF
      ENDDO

      END FUNCTION feasible


! Objective function
      DOUBLEPRECISION FUNCTION f(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,j
      LOGICAL:: indicator

      DO i=1,pointsCalculated
          indicator = .TRUE.
          DO j=1,n
              IF (table(i,j) .NE. x(j)) THEN
                  indicator = .FALSE.
                  EXIT
              ENDIF
          ENDDO

          IF (indicator) THEN
              f = table(i,n+1)
              RETURN
          ENDIF
      ENDDO

      f=0.0d0
      DO i=1,n-1
         f=f+(100.0d0*(x(i+1)-x(i)**2)**2+(1.0d0-x(i))**2)
      ENDDO

      table(pointsCalculated+1,1:n) = x(:)
      table(pointsCalculated+1,n+1) = f

      pointsCalculated = pointsCalculated + 1
      countf=countf+1

      END FUNCTION f


! Filled function
      DOUBLEPRECISION FUNCTION p(x,xstar,fx,fxstar)
      USE constants
      IMPLICIT NONE
      INTEGER::i,s
      INTEGER,INTENT(IN)::x(n),xstar(n)
      DOUBLEPRECISION,INTENT(IN)::fx,fxstar
      DOUBLEPRECISION::f,y,v,a

      y=fx-fxstar
```

```
      s=0
      DO i=1,n
         s=s+(x(i)-xstar(i))**2
      ENDDO

      v=mu*((1.0d0-c)*((1.0d0-c*mu)/(mu-c*mu))**(-y/tao)+c)

      a=y*v
      p=a*y-rho*s
      countff=countff+1

      END FUNCTION p


! Define the LOGICAL FUNCTION to check if a vertex exists
      LOGICAL FUNCTION vertex(x)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x(n)
      INTEGER::i,tally

      tally=0

      DO i=1,n
         IF (x(i).EQ.xl(i).OR.x(i).EQ.xu(i)) THEN
             tally=tally+1
         ENDIF
      ENDDO

      IF (tally.EQ.n) THEN
          vertex=.TRUE.
      ELSE
          vertex=.FALSE.
      ENDIF

      END FUNCTION vertex


! Local search of the original function,f
      SUBROUTINE minf(x0,xstar,fxbest)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN)::x0(n)
      INTEGER,INTENT(OUT)::xstar(n)
      DOUBLEPRECISION,INTENT(OUT)::fxbest
      INTEGER::x(n),xbest(n),neighbour(n),i,j
      DOUBLEPRECISION::f,fx,temp
      LOGICAL::feasible

      x(:)=x0(:)

      DO

        fx=f(x)
        xbest(:)=x(:)
        fxbest=fx

        DO j=1,2*n
           neighbour(:)=x(:)+e(:,j)

           IF (feasible(neighbour)) THEN
               temp=f(neighbour)
               IF (temp.LT.fxbest) THEN
                   xbest(:)=neighbour(:)
                   fxbest=temp
               ENDIF
```

```
          ENDIF
        ENDDO

        IF (fxbest.EQ.fx) THEN
            xstar(:)=x(:)
            RETURN
        ELSE
            x(:)=xbest(:)
        ENDIF

      ENDDO

      END SUBROUTINE minf


! Local search of the filled function.
      SUBROUTINE minp(xstar,newx,flag)
      USE constants
      IMPLICIT NONE
      INTEGER,INTENT(IN):: xstar(n)
      INTEGER,INTENT(OUT):: newx(n),flag
      INTEGER:: x(n),i,j,FLOOR,timeArray(3),r(n,m)
      DOUBLEPRECISION:: fxstar,f,fx,RAND
      LOGICAL:: feasible
      EXTERNAL::minp1

! Generate a set of random points from FORTRAN.
      CALL itime(timeArray)
      DO j=1,m
         DO i=1,n
            r(i,j)=FLOOR(xl(i)+(RAND(timeArray(3)+j+i))*(xu(i)-xl(i)+1))
         ENDDO
      ENDDO

      fxstar = f(xstar)

      DO j=1,m
         IF (feasible(r(:,j))) THEN
             fx=f(r(:,j))
             PRINT*,"feasible random point=",r(:,j),"and f=",fx
             IF (fx .LT. fxstar) THEN
                 flag = 1
                 newx(:) = r(:,j)
                 RETURN
             ENDIF
         ENDIF
      ENDDO

      DO

        IF (rho .LT. rhol) THEN
            flag = 2
            RETURN
        ENDIF

        DO j=1,m
            x(:) = xstar(:) + e(:,j)
            IF (feasible(x)) THEN
                CALL minp1(xstar,fxstar,x,flag)
                IF (flag .EQ. 1) THEN
                    newx(:) = x(:)
                    RETURN
                ENDIF
            ENDIF
        ENDDO

        rho = rho*rhohat
```

```
ENDDO

END SUBROUTINE minp


SUBROUTINE minp1(xstar,fxstar,x,flag)
USE constants
IMPLICIT NONE
INTEGER,INTENT(IN):: xstar(n)
DOUBLEPRECISION,INTENT(IN):: fxstar
INTEGER,INTENT(INOUT):: x(n)
INTEGER,INTENT(OUT):: flag
INTEGER::xbest(n),neighbour(n),betterxbest(n),j,flag1
DOUBLEPRECISION:: f,fx,fcurrent,fn,p,pcurrent,pbest,temp,total,totalbest
LOGICAL::feasible,vertex

flag = 0

DO

   xbest(:) = x(:)
   fcurrent = f(x)
   pcurrent = p(x,xstar,fcurrent,fxstar)
   pbest = pcurrent

   flag1 = 0

   DO j=1,2*n
       neighbour(:) = x(:) + e(:,j)

       IF (feasible(neighbour)) THEN
           fn = f(neighbour)
           IF (fn .LT. fxstar) THEN
               flag = 1
               x(:) = neighbour(:)
               RETURN
           ELSE
               temp = p(neighbour,xstar,fn,fxstar)
               IF (temp .LT. pbest) THEN
                    xbest(:) = neighbour(:)
                    pbest = temp
               ENDIF

               IF (temp .LT. pcurrent .AND. fn .LT. fcurrent) THEN
                   IF (flag1 .eq. 0) THEN
                       betterxbest(:) = neighbour(:)
                       totalbest = temp + fn
                       flag1 = 1
                   ELSE
                       total = temp + fn
                       IF (total .LT. totalbest) THEN
                           totalbest = total
                           betterxbest(:) = neighbour(:)
                       ENDIF
                   ENDIF
               ENDIF
           ENDIF
       ENDIF

   ENDDO

   IF (pbest .EQ. pcurrent) THEN
       IF (vertex(x)) THEN
           RETURN
       ELSE
           mu = mu*muhat
```

```
            CYCLE
        ENDIF
    ELSE
        IF (flag1 .EQ. 1) THEN
            x(:) = betterxbest(:)
        ELSE
            x(:) = xbest(:)
        ENDIF
    ENDIF

ENDDO

END SUBROUTINE minp1
```

Bibliography

[1] N. U. Ahmed. *Linear and nonlinear filtering for scientists and engineers*. Singapore: World Scientific, 1998.

[2] M. Ashari. *Optimisation of photovoltaic/diesel/battery hybrid power systems for remote area electrification*. Master's thesis, Department of Electrical and Computer Engineering, Curtin University of Technology, 1997.

[3] J. Bang-Jensen, G. Gutin, and A. Yeo. When the greedy algorithm fails. *Discrete Optimization*, 1:121–127, 2004.

[4] J. S. Baras and A. Bensoussan. Optimal sensor scheduling in nonlinear filtering of diffusion process. *SIAM Journal of Control and Optimization*, 27:786–813, 1989.

[5] C. D. Barley, L. T. Flowers, P. J. Benavidez, R. L. Abergas, and R. B. Barruela. Feasibility of hybrid retrofits to off-grid diesel power plants in the Philippines. *Prepared for Windpower'99, Burlington, Vermont*, June 20-23, 1999.

[6] R. E. Bellman. *Dynamic programming*. Princeton: Princeton University Press, 1957.

[7] H. Binder, T. Cronin, P. Lundsager, J. F. Manwell, U. Abdulwahid, and I. Baring-Gould. *Lifetime modelling of lead acid batteries*. Roskilde, Denmark: Riso National Laboratory, 2005.

[8] C. Blum and A. Roli. Metaheuristics in combinatorial optimization:

Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[9] W. Bosarge Jr, O. Johnson, I. B. M. S. Center, and T. X. Houston. Direct method approximation to the state regulator control problem using a Ritz-Trefftz suboptimal control. *IEEE Transactions on Automatic Control*, 15(6):627–631, 1970.

[10] E. K. Burke and G. Kendall. *Search methodology: Introductory tutorials in optimization and decision support techniques*. New York: Springer, 2005.

[11] L. Caccetta, I. Loosen, and V. Rehbock. Computational aspects of the optimal transit path problem. *Journal of Industrial and Management Optimization*, 4(1):95–105, 2008.

[12] J. Cai and G. Thierauf. Evolution strategies for solving discrete optimization problems. *Advances in Engineering Software*, 25:177–183, 1996.

[13] E. F. Carrasco and J. R. Banga. A hybrid method for the optimal control of chemical process. In *UKACC International Conference on Control 98, University of Wales, Swansea*, 455, pages 925–930, September 1-4, 1998.

[14] D. Castanon and L. Carin. *Stochastic control theory for sensor management. In A. Hero and D. Castanon and D. Cochran and K. Kastella (Eds.), Foundations & Applications of Sensor Management*. Boston: SpringerLink, 7–32, 2008.

[15] D. Chakrabarty, N. R. Devanur, and V. V. Vazirani. *New geometry-inspired relaxations and algorithms for the Metric Steiner Tree Problem. In A. Lodi and A. Panconesi and G. Rinaldi (Eds.), Integer Programming and Combinatorial Optimization: 13th International Conference, IPCO 2008 Bertinoro, Italy, May 26-28, 2008 Proceedings*. New York: Springer-Verlag, 344–358, 2008.

[16] T. H. Chung, V. Gupta, B. Hassibi, J. Burdick, and R. M. Murray. Scheduling for distributed sensor networks with single sensor measurement per time step. In *IEEE International Conference on Robotics & Automation, New Orleans*, pages 187–192, April 26 - May 1, 2004.

[17] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[18] P. Deuflhard. A modified Newton method for the solution of ill-conditioned systems of nonlinear equations with application to multiple shooting. *Numerische Mathematik*, 22(4):289–315, 1974.

[19] L. C. W. Dixon and G. P. Szego, editors. *Towards Global Optimization*. North-Holland: Elservier Science Limited, 1975.

[20] G. Dobson. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Mathematics of Operations Research*, 7(4):515–531, 1982.

[21] N. J. Edwards and C. J. Goh. Direct training method for a continuous-time nonlinear optimal feedback controller. *Journal of Optimization Theory and Applications*, 84(3):509–528, 1995.

[22] M. Egerstedt, Y. Wardi, and F. Delmotte. Optimal control of switching times in switched dynamical systems. In *Proceedings of the 42nd IEEE Conference on Decision and Control, Maui, Hawaii*, volume 3, pages 2138–2143, December 9-12, 2003.

[23] A. Etzel. Mixed discrete optimization of multiple-valued systems. In *27th International Symposium on Multiple-Valued Logic Proceedings, Antigonish, Canada*, pages 259–264, May 28-30, 1997.

[24] A. Federgruen and H. Groenevelt. The greedy procedure for resource allocation problems: Necessary and sufficient conditions for optimality. *Operations Research*, 34(6):909–918, 1986.

[25] Z. G. Feng and K. L. Teo. A discrete filled function method for the design of FIR filters with signed-powers-of-two coefficients. *IEEE Transactions on Signal Processing*, 56(1):134–139, 2008.

[26] Z. G. Feng, K. L. Teo, and V. Rehbock. Optimal sensor scheduling in continuous time. *Dynamic Systems and Applications*, 17:331–350, 2008.

[27] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 50(12):1861–1871, 2004.

[28] R. Fletcher. *Practical methods of optimization*. Chichester: Wiley, 1987.

[29] R. Fukasawa and M. Goycoolea. *On the exact separation of mixed integer knapsack cuts. In M. Fischetti and D. P. Williamson (Eds.), Integer Programming and Combinatorial Optimization: 12th International Conference, IPCO 2007 Ithaca, New York, June 25-27, 2007 Proceedings*. New York: Springer-Verlag, 225–239, 2007.

[30] R. Ge. A filled function method for finding a global minimizer of a function of several variables. *Mathematical Programming*, 46(1):191–204, 1990.

[31] R. Ge and C. Huang. A continuous approach to nonlinear integer programming. *Applied Mathematics and Computation*, 34(1):39–60, 1989.

[32] A. M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical Programming Study*, 2:82–114, 1974.

[33] P. E. Gill, W. Murray, A. Saunders, and M. H. Wright. *User's guide for NPSOL 5.0: A FORTRAN package for nonlinear programming*. Stanford University, http://cam.ucsd.edu/ peg/papers/npdoc.pdf, 2001.

[34] F. Glover. *Tabu search and adaptive memory programming–Advances, applications and challenges. In R. S. Barr, R. V. Helgason, J. L. Kennington (Eds.), Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*. Norwell, MA: Kluwer Academic Publishers, 1–75, 1996.

[35] C. J. Goh and K. L. Teo. Control parametrization: A unified approach to optimal control problem with general constraints. *Automatica*, 24(1):3–18, 1988.

[36] C. J. Goh and K. L. Teo. MISER: A FORTRAN program for solving optimal control problems. *Advances in Engineering Software*, 10(2):90–99, 1988.

[37] A. A. Goldstein and J. F. Price. On descent from local minima. *Mathematics of Computation*, 25(115):569–574, 1971.

[38] Y. H. Gu and Z. Y. Wu. A new filled function method for nonlinear integer programming problem. *Applied Mathematics and Computation*, 173(2):938–950, 2006.

[39] M. R. Guide. The MathWorks. *Inc., Natick, MA, http://www.mathworks.com/*, 1998.

[40] O. Günlük and J. Linderoth. *Perspective relaxation of mixed integer nonlinear programs with indicator variables. In A. Lodi and A. Panconesi and G. Rinaldi (Eds.), Integer Programming and Combinatorial Optimization: 13th International Conference, IPCO 2008 Bertinoro, Italy, May 26-28, 2008 Proceedings*. New York: Springer-Verlag, 1–16, 2008.

[41] O. K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546, 1985.

[42] G. A. Hicks and W. H. Ray. Approximation methods for optimal control systems. *Canadian Journal of Chemical Engineering*, 49(4):522–528, 1971.

[43] F. S. Hillier and Lieberman G. J. *Introduction to Operations Research*. New York: McGraw-Hill, 2005.

[44] W. Hock and K. Schittkowski. *Test examples for nonlinear programming codes*. New York: Springer-Verlag, 1980.

[45] S. A Hovanessian. *Introduction to sensor systems*. Norwood, MA: Artech House, Inc., 1988.

[46] P. Howlett. Optimal strategies for the control of a train. *Automatica*, 32(4):519–532, 1996.

[47] D. H. Jacobson and D. Q. Mayne. *Differential dynamic programming*. New York: Elsevier Publishing Company, 1970.

[48] L. S. Jennings, M. E. Fisher, K. L. Teo, and C. J. Goh. *MISER3.3–Optimal control software: Theory and user manual*. University of Western Australia, http://www.maths.uwa.edu.au/˜les/MISER3.3/ch1.pdf, 2004.

[49] L. S. Jennings and K. L. Teo. A numerical algorithm for constrained optimal control problems with applications to harvesting. *Dynamics of Complex Interconnected Biological Systems*, pages 218–234, 1990.

[50] M. I. Kamien and N. L. Schwartz. *Dynamic optimization: The calculus of variations and optimal control in economics and management*. North-Holland: Elsevier Science, 1991.

[51] C. Y. Kaya and J. L. Noakes. Computations and time-optimal controls. *Optimal Control Applications and Methods*, 17(3):171–185, 1996.

[52] C. Y. Kaya and J. L. Noakes. A global control law with implications in time-optimal control. In *Proceedings of the 33rd IEEE Conference on Decision and Control, Orlando, Florida*, pages 3823–3824, Dec 14-16, 1994.

[53] C. Y. Kaya and J. L. Noakes. The leap-frog algorithm and optimal control: Background and demonstration. In *Proceedings of International Conference on Optimization Techniques and Applications (ICOTA'98), Perth*, pages 835–842, Jul 1-3, 1998.

[54] C. Y. Kaya and J. L. Noakes. The leap-frog algorithm and optimal control: Theoretical aspects. In *Proceedings of International Conference on Opti-*

mization Techniques and Applications (ICOTA'98), Perth, pages 843–850, Jul 1-3, 1998.

[55] E. Khmelnitsky. A combinatorial, graph-based solution method for a class of continuous time optimal control problems. *Mathematics of Operations Research*, 27(2):312–325, 2002.

[56] R. Lakshmanan and G. Stephanopoulos. Synthesis of operating procedures for complete chemical plants-II: A nonlinear planning methodology. *Computers in Chemical Engineering*, 12(9/10):1003–1021, 1988.

[57] T. Lambert, P. Gilman, and P. Lilienthal. *Micropower system modeling with HOMER. In Farret, F. A. and Simoes, M. G. (Eds.), Integration of Alternative Sources of Energy*. New York: John Wiley & Sons, Inc., 2006.

[58] H. W. J. Lee, M. M. Ali, and K. H. Wong. Global optimization for a class of optimal discrete-valued control problem. *Dynamics of Continuous, Discrete and Impulsive Systems, Series B*, 11(6):735–756, 2004.

[59] H. W. J. Lee, X. Q. Cai, and K. L. Teo. Control parametrization enhancing technique for time optimal control problems. *Mathematical Problems in Engineeering*, 7:155–175, 2001.

[60] H. W. J. Lee, K. L. Teo, and X. Q. Cai. An optimal control approach to nonlinear mixed integer programming problems. *Computers and Mathematics with Applications*, 36(3):87–105, 1998.

[61] H. W. J. Lee, K. L. Teo, L. S. Jennings, and V. Rehbock. Control parametrization enhancing technique for time optimal control problems. *Dynamic Systems and Applications*, 6:243–262, 1997.

[62] H. W. J. Lee, K. L. Teo, W. R. Lee, and S. Wang. Construction of suboptimal feedback control for chaotic systems using B-splines with optimally chosen knot points. *International Journal of Bifurcation and Chaosin in Applied Sciences and Engineering*, 11(9):2375–2387, 2001.

[63] H. W. J. Lee, K. L. Teo, and A. E. B. Lim. Sensor scheduling in continuous time. *Automatica*, 37(12):2017–2023, 2001.

[64] H. W. J. Lee, K. L. Teo, V. Rehbock, and L. S. Jennings. Control parametrization enhancing technique for optimal discrete-valued control problems. *Automatica*, 35(8):1401–1407, 1999.

[65] W. J. Lee, A. V. Cabot, and M. A. Venkataramanan. A branch and bound algorithm for solving separable convex integer programming problems. *Computers and Operations Research*, 21(9):1011–1024, 1994.

[66] W. R. Lee, V. Rehbock, L. Caccetta, and K. L. Teo. Numerical solution of optimal control problems with discrete-valued system parameters. *Journal of Global Optimization*, 23:233–244, 2002.

[67] C. C. Lim and K. L. Teo. Optimal insulin infusion control via a mathematical blood glucoregulatory model with fuzzy parameters. *Cybernetics and Systems*, 22(1):1–16, 1991.

[68] Y. Liu, K. L. Teo, L. S. Jennings, and S. Wang. On a class of optimal control problems with state jumps. *Journal of Optimization Theory and Applications*, 98(1):65–82, 1998.

[69] R. Loxton. *A global computational approach to a class of optimal control problems*. B.Sc. Honours Project, Department of Mathematics and Statistics, Curtin University of Technology, 2006.

[70] R. C. Loxton, K. L. Teo, V. Rehbock, and W. K. Ling. Optimal switching instants for a switched-capacitor DC/DC power converter. *Automatica*, 45(4):973–980, 2009.

[71] S. K. Lucas and C. Y. Kaya. Switching-time computation for bang-bang control laws. In *Proceedings of the American Control Conference 2001, Arlington, Virginia*, volume 1, pages 176–181, Jun 25-27, 2001.

[72] R. Luus. Optimal control by dynamic programming using accessible grid points and region contraction. *Hungarian Journal of Industrial Chemistry*, 17:523–543, 1989.

[73] R. Luus. Application of dynamic programming to high-dimensional non-linear optimal control problems. *International Journal of Control*, 52(1):239–250, 1990.

[74] R. Luus. Optimal control by dynamic programming using systematic reduction in grid size. *International Journal of Control*, 51(5):995–1013, 1990.

[75] R. Luus. Piecewise linear continuous optimal control by iterative dynamic programming. *Industrial & Engineering Chemistry Research*, 32(5):859–865, 1993.

[76] R. Luus. Optimal control of batch reactors by iterative dynamic programming. *Journal of Process Control*, 4(4):218–226, 1994.

[77] R. Luus. Determination of the region sizes for LJ optimization procedure. *Hungarian Journal of Industrial Chemistry*, 26:281–286, 1998.

[78] R. Luus. *Iterative dynamic programming*. Boca Raton: Chapman & Hall, 2000.

[79] R. Luus. Toward global optimization of a difficult optimal control problem. *Hungarian Journal of Industrial Chemistry*, 29(1):61–66, 2001.

[80] R. Luus. Use of Luus-Jaakola optimization procedure for singular optimal control problems. *Nonlinear Analysis*, 47(8):5647–5658, 2001.

[81] R. Luus. Use of Luus-Jaakola optimization procedure with flexible stage lengths for optimal control. In *Proceedings of IASTED International Conference on Intelligent Systems and Control, Santa Barbara, California*, pages 402–405, Oct 28-30, 1999.

[82] R. Luus and V. Dong. Use of iterative dynamic programming in optimization of reactor systems. In *Proceedings of the IFAC Symposium on Dynamics and Control of Chemical Reactors, Distillation Columns and Batch Processes, Helsingor, Denmark*, pages 45–49, June 7-9, 1995.

[83] R. Luus, F. Hartig, and F. J. Keil. Optimal drug scheduling of cancer chemotherapy by direct search optimization. *Hungarian Journal of Industrial Chemistry*, 23:55–58, 1995.

[84] R. Luus and D. Hennessy. Optimization of fed-batch reactors by the Luus-Jaakola optimization procedure. *Industrial Engineering and Chemistry Research*, 38(5):1948–1955, 1999.

[85] R. Luus and T. H. I. Jaakola. Optimization by direct search and systematic reduction of the size of search region. *American Institute of Chemical Engineers Journal*, 19(4):760–766, 1973.

[86] R. Luus and O. Rosen. Application of dynamic programming to final state constrained optimal control problems. *Industrial & Engineering Chemistry Research*, 30(7):1525–1530, 1991.

[87] K. Madsen and J. Žilinskas. *Testing branch-and-bound methods for global optimization*. IMM, Department of Mathematical Modelling, Technical University of Denmark, 2000.

[88] J. F. Manwell and J. G. McGowan. Lead acid battery storage model for hybrid energy systems. *Solar Energy*, 50(5):399–405, 1993.

[89] J. F. Manwell, A. Rogers, G. Hayman, C. T. Avelar, J. G. McGowan, U. Abdulwahid, and K. Wu. *HYBRID2 - A hybrid system simulation model theory manual*. Boston: Renewable Energy Research Laboratory, Department of Mechanical Engineering, University of Massachusetts, June 30, 2006.

[90] R. E. Marsten and T. L. Morin. A hybrid approach to discrete mathematical programming. *Mathematical Programming*, 14:21–40, 1978.

[91] R. K. Maskey and F. Nestmann. Hydro based renewable hybrid power system for rural electrification: A concept paper. *Retrived June 12, 2008, from http://www.mtnforum.org/apmn/hybridconcept.htm*, 2008.

[92] H. Maurer and H. J. Oberle. Second order sufficient conditions for optimal control problems with free final time: The Riccati approach. *SIAM Journal on Control and Optimization*, 41(2):380–403, 2002.

[93] H. Maurer and N. P. Osmolovskii. Second order sufficient conditions for time-optimal bang-bang control. *SIAM Journal on Control and Optimization*, 42(6):2239–2263, 2004.

[94] H. Maurer and S. Pickenhain. Second-order sufficient conditions for control problems with mixed control-state constraints. *Journal of Optimization Theory and Applications*, 86(3):649–667, 1995.

[95] L. Michel and P. Van Hentenryck. A simple tabu search for warehouse location. *European Journal of Operational Research*, 157:576–591, 2004.

[96] A. Miele. Gradient algorithms for the optimization of dynamic systems. *Control and Dynamic Systems: Advances in Theory and Applications*, 16:1–52, 1980.

[97] A. Miele and T. Wang. Dual properties of sequential gradient-restoration algorithms for optimal control problems. *Optimization and Related Fields*, pages 331–357, 1986.

[98] A. Miele and T. Wang. Primal-dual properties of sequential gradient-restoration algorithms for optimal control problems, Part 2: General problems. *Journal of Mathematical Analysis and Applications*, 119(1-2):21–54, 1986.

[99] A. Miele, T. Wang, and V. K. Basapur. Primal and dual properties formulations of sequential gradient-restoration algorithms for trajectory optimization problems. *Acta Astronautica*, 13:491–505, 1986.

[100] M. B. Milam, K. Mushambi, and R. M. Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *Proceedings of the 39th IEEE Conference on Decision and Control, Sydney, Australia*, pages 845–851, Dec 12-15, 2000.

[101] C. Mohan and H. T. Nguyen. A controlled random search technique incorporating the simulated annealing concept for solving integer and mixed integer global optimization problems. *Computational Optimization and Applications*, 14:103–132, 1999.

[102] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

[103] D. D. Morrison, J. D. Riley, and J. F. Zancanaro. Multiple shooting method for two-point boundary value problems. *Communications of the ACM*, 5(12):613–614, 1962.

[104] S. H. Musick. *Defence applications. In A. Hero and D. Castanon and D. Cochran and K. Kastella (Eds.), Foundations & applications of sensor management*. Boston: SpringerLink, 257-268, 2008.

[105] C. K. Ng, D. Li, and L. S. Zhang. *Filled function approaches to nonlinear integer programming: A survey. In S. H. Hou and X. M. Yang and G. Y. Chen (Eds.), Frontiers in Optimization and Control, vol. 20*. Norwell, MA: Kluwer Academic Publishers, 1-20, 2005.

[106] C. K. Ng, D. Li, and L. S. Zhang. Discrete global descent method for discrete global optimization and nonlinear integer programming. *Journal of Global Optimization*, 37(3):357–379, 2007.

[107] C. K. Ng, L. S. Zhang, D. Li, and W. W. Tian. Discrete filled function method for discrete global optimization. *Computational Optimization & Applications*, 31(1):87–115, 2005.

[108] L. Noakes. A global algorithm for geodesics. *Journal of the Australian Mathematical Society-Series A*, 64(1):37–50, 1998.

[109] H. J. Oberle and B. Sothmann. Numerical computation of optimal feed rates for a fed-batch fermentation model. *Journal of Optimization Theory and Applications*, 100(1):1–13, 1999.

[110] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The mathematical theory of optimal processes*. New York: Wiley Interscience, 1962.

[111] V. Rehbock and L. Caccetta. Two defence applications involving discrete-valued optimal control. *Journal of ANZIAM*, 44(E):E33–E54, 2002.

[112] V. Rehbock and P. Souksomvang. Optimal reset time for hybrid power systems. In *Proceedings of the 18th National ASOR Conference & 11th Australian Optimisation Day, Perth, Australia*, pages 229–236, September, 26-28 2005.

[113] M. Rim and R. Jain. Lower-bound performance estimation for the high-level synthesis scheduling problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):451–458, 1994.

[114] M. Ronen, Y. Shabtai, and H. Guterman. Optimization of feeding profile for a fed-batch bioreactor by an evolutionary algorithm. *Journal of Biotechnology*, 97(3):253–263, 2002.

[115] S. L. Rosen and C. M. Harmonosky. An improved simulated annealing simulation optimization method for discrete parameter stochastic systems. *Computers and Operations Research*, 32:343–358, 2005.

[116] T. Ruby, V. Rehbock, and W. B. Lawrence. Optimal control of hybrid power systems. *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Applications & Algorithms*, 10:429–439, 2003.

[117] A. Rusnák, M. Fikar, M. A. Latifi, and A. Mészáros. Receding horizon iterative dynamic programming with discrete time models. *Computers and Chemical Engineering*, 25(1):161–167, 2001.

[118] K. Schittkowski. On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search function. *Optimization: A Journal of Mathematical Programming and Operations Research*, 14(2):197–216, 1983.

[119] K. Schittkowski. NLPQL: A FORTRAN subroutine solving constrained nonlinear programming problems. *Annals of Operations Research*, 5(1):485–500, 1986.

[120] K. Schittkowski. *More test examples for nonlinear programming codes*. New York: Springer-Verlag, 1987.

[121] A. Schwartz and E. Polak. Consistent approximations for optimal control problems based on Runge-Kutta integration. *SIAM Journal on Control and Optimization*, 34(4):1235–1269, 1996.

[122] A. L. Schwartz. *Theory and implementation of numerical methods based on Runge-Kutta integration for solving optimal control problems*. Ph.D. thesis, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 1997.

[123] A. L. Schwartz and E. Polak. *RIOTS: A Matlab toolbox for solving optimal control problems, version 1.0*, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 1996.

[124] G. Seeling-Hochmuth. *Optimisation of hybrid energy systems sizing and operation control*. Ph.D. thesis, University of Kassel, 1998.

[125] G. C. Seeling-Hochmuth. A combined optimisation concept for the design and operation strategy of hybrid-PV energy systems. *Solar Energy*, 61:77–87, 1997.

[126] M. S. Shaikh and P. E. Caines. On the optimal control of hybrid systems: Analysis and zonal algorithms for trajectory and schedule optimization. In *Proceedings of the 42nd IEEE Conference on Decision and Control, Maui, Hawaii*, volume 3, pages 2144–2149, December 9-12, 2003.

[127] Y. Shang and L. Zhang. A filled function method for finding a global minimizer on global integer optimization. *Journal of Computational and Applied Mathematics*, 181(1):200–210, 2005.

[128] Y. Shang and L. Zhang. Finding discrete global minima with a filled function for integer programming. *European Journal of Operational Research*, 189(1):31–40, 2008.

[129] A. Siburian. *Numerical methods for robust, singular and discrete valued optimal control problems*. Ph.D. thesis, Department of Mathematics and Statistics, Curtin University of Technology, 2003.

[130] H. R. Sirisena. Computation of optimal control using a piecewise polynomial parameterization. *IEEE Transactions on Automatic Control*, 18(4):409–411, 1973.

[131] H. R. Sirisena and F. S. Chou. An efficient algorithm for solving optimal control problems with linear terminal constraints. *IEEE Transactions on Automatic Control*, 21(2):275–277, 1976.

[132] H. R. Sirisena and F. S. Chou. Convergence of control parametrization Ritz method for nonlinear optimal control problems. *Journal of Optimization Theory and Applications*, 29(3):369–382, 1979.

[133] H. R. Sirisena and K. S. Tan. Computation of constrained optimal control using parameterization techniques. *IEEE Transactions on Automatic Control*, 19(4):431–433, 1974.

[134] D. E. Stewart. A numerical algorithm for optimal control problems with switching costs. *The ANZIAM Journal*, 34(02):212–228, 1992.

[135] V. Tassone and R. Luus. Reduction of allowable values for control in iterative dynamic programming. *Chemical Engineering Science*, 48(22):3864–3868, 1993.

[136] K. L. Teo and D. J. Clements. A control parametrization algorithm for convex optimal control problems with linear constraints. *Numerical Functional Analysis and Optimization*, 8:515–540, 1986.

[137] K. L. Teo and C. J. Goh. A computational method for combined optimal parameter selection and optimal control problems with general constraints. *Journal of Mathematical Society, Series B*, 30:350–364, 1989.

[138] K. L. Teo and C. J. Goh. Computational techniques for optimal relaxed control problems. *Journal of Optimization Theory and Applications*, 60(1):117–133, 1989.

[139] K. L. Teo, C. J. Goh, and C. C. Lim. A computational method for a class of dynaical optimization problems in which the terminal time is conditionally free. *Journal of Mathematical Control and Information*, 6(1):81–95, 1989.

[140] K. L. Teo, C. J. Goh, and K. H. Wong. *A unified computational approach to optimal control problems*. Essex: Longman Scientific & Technical, 1991.

[141] K. L. Teo and L. S. Jennings. Nonlinear optimal control problems with continuous state inequality constraints. *Journal of Optimization Theory and Applications*, 63(1):1–22, 1989.

[142] K. L. Teo and L. S. Jennings. Optimal control with a cost on changing control. *Journal of Optimization Theory and Applications*, 68(2):335–357, 1991.

[143] K. L. Teo, L. S. Jennings, H. W. J. Lee, and V. Rehbock. The control parametrization enhancing transform for constrained optimal control problems. *Journal of Australian Mathematical Society, Series B*, 40:341–335, 1998.

[144] K. L. Teo, K. K. Leong, and C. J. Goh. Nonlinearly constrained optimal control problems involving piecewise smooth control. *Journal of Mathematical Society, Series B*, 32:151–179, 1990.

[145] K. L. Teo and V. Rehbock. Applied and computational optimal control, working book.

[146] R. Tiryono, W. R. Lee, and V. Rehbock. Optimal design and operation of hybrid power systems. In *Proceedings of Industrial Optimization Symposium, Perth, Australia*, pages 324–335, August 23-25, 2004.

[147] N. Turkkan. Discrete optimization of structures using a floating-point genetic algorithm. In *Annual Conference of the Canadian Society for Civil Engineering, Moncton, Canada*, June 4-7, 2003.

[148] O. Von Stryk. *Numerical solution of optimal control problems by direct collocation. In R. Bulirsch and A. Miele and J. Stoer and K. H. Well (Eds.), Optimal Control - Calculus of Variations, Optimal Control Theory and Numerical Methods, International Series in Numerical Mathematics 111*. Basel: Birkhäuser, 129–143, 1993.

[149] O. Von Stryk. Users guide for DIRCOL 2.1: A direct collocation method for the numerical solution of optimal control problems. *Technische Universität Darmstadt*, 1999.

[150] B. Wichert. PV-diesel hybrid energy systems for remote area power generation–A review of current practice and future developments. *Renewable and Sustainable Energy Reviews*, 1(3):209–228, 1997.

[151] B. Wichert and W. B. Lawrance. Photovoltaic resource and load demand forecasting in stand-alone renewable energy systems. In *2nd World Conference on Photovoltaic Solar Energy conversion, Vienna*, volume 3, pages 3194–3197, July 6-10, 1998.

[152] K. H. Wong, D. J. Clements, and K. L. Teo. Optimal control computation for nonlinear time-lag systems. *Journal of Optimization Theory and Applications*, 47(1):91–107, 1985.

[153] C. Z. Wu, K. L. Teo, and V. Rehbock. A filled function method for optimal discrete-valued control problems. *Journal of Global Optimization*, 44(2):213–225, 2009.

[154] S. J. Wu and P. T. Chow. Steady-state genetic algorithms for discrete optimization of trusses. *Computers and Structures*, 56(6):979–991, 1995.

[155] Z. Wu and B. W. Wah. *The theory of discrete Lagrange multipliers for nonlinear discrete optimization. In J. Jaffar(Ed.), Principles and Practice of Constraint Programming: 5th International Conference, CP 1999, Alexandria, Virginia, October 11-14, 1999 Proceedings*. New York: Springer-Verlag, 28–42, 1999.

[156] Z. Wu and B. W. Wah. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In *Proceeding of the 17th National Conference on Artificial Intelligence, Austin, Texas*, pages 310–315, July 30 - August 3, 2000.

[157] X. Xu and P. J. Antsaklis. Optimal control of switched autonomous systems. In *Proceedings of the 41st IEEE Conference on Decision and Control, Las Vegas, Nevada*, volume 4, pages 4401–4406, December 10-13, 2002.

[158] Y. Yamashita and M. Shima. Numerical computational method using genetic algorithm for the optimal control problem with terminal constraints and free parameters. *Nonlinear Analysis*, 30(4):2285–2290, 1997.

[159] X. Q. Yang and C. J. Goh. *A nonlinear Lagrangian function for discrete optimization problems. In A. Migdalas, A. Pardalos, P. Varbrand, K. Holmqvist (Eds.), From Local to Global Optimization*. The Netherlands: Kluwer Academic Publishers, 291–304, 2000.

[160] Y. Yang and Y. Liang. A new discrete filled function algorithm for discrete global optimization. *Journal of Computational and Applied Mathematics*, 202(2):280–291, 2007.

[161] Y. Yang, Z. Wu, and F. Bai. A filled function method for constrained nonlinear integer programming. *Journal of Industrial and Management Optimization*, 4(2):353–362, 2008.

[162] Y. Yang and L. Zhang. A gradually descent method for discrete global optimization. *Journal of Shanghai University (English Edition)*, 11(1):39–44, 2007.

[163] A. Zanette, M. Fischetti, and E. Balas. *Can pure cutting plane algorithms work ? In A. Lodi and A. Panconesi and G. Rinaldi (Eds.), Integer Programming and Combinatorial Optimization: 13th International Conference, IPCO 2008 Bertinoro, Italy, May 26-28, 2008 Proceedings*. New York: Springer-Verlag, 416–434, 2008.

[164] J. L. Zhou, A. L. Tis, and C. T. Lawrence. *User's guide for FFSQP version 3.7: A FORTRAN code for solving constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality and linear constraints*. University of Maryland, http://www.aemdesign.com/download-ffsqp/ffsqp-manual.pdf, 1997.

[165] W. Zhu. An approximate algorithm for nonlinear integer programming. *Applied Mathematics and Computations*, 93(2-3):183–193, 1998.