

https://dspace.org/

An efficient algorithm for nonlinear integer programming

Moepya, Stephen Obakeng

2011-11-02
http://hdl.handle.net/10539/10679

Downloaded from a DSpace repository. https://dspace.org/

An efficient algorithm for nonlinear integer programming

Stephen Obakeng Moepya



A Dissertation submitted for the degree of
Masters of Science in
the field of global optimization.

School of Computational and Applied Mathematics,
University of the Witwatersrand,
Johannesburg, South Africa.

July 4, 2011

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science to the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other university.

_____

July 4, 2011

Abstract

This dissertation is concerned with discrete global optimization of nonlinear problems. These problems are constrained and unconstrained and are not easily solvable since there exists multiplicity of local and global minima. In this dissertation, we study the current methods for solving such problems and highlight their inefficiencies. We introduce a new local search procedure. We study the rapidly-exploring random tree (RRT) method, found mostly in the research area of robotics. We then design two global optimization algorithms based on RRT. RRT has never been used in the field of global optimization. We exploit its attractive properties to develop two new algorithms for solving the discrete nonlinear optimization problems. The first method is called RRT-Optimizer and is denoted as RRTOpt. RRTOpt is then modified to include probabilistic elements within the RRT. We have denoted this method by RRTOptv1. Results are generated for both methods and numerical comparisons are made with a number of recent methods.

Keywords: Global optimization, nonlinear integer programming, local search, multi-start, rapidly-exploring random trees.

Acknowledgements

First and foremost, I give my whole-hearted acknowledgment to my **Heavenly Father**, the Lord of lords, ruler of my universe, Alpha and Omega. He has shone a light unto my path and has showered me with countless blessings. Without him I am nothing.

In reference to my academic endeavors, I would like to extend my sincere appreciation and gratitude to my supervisor, Prof. M.M Ali. He has put a tremendous amount of time and effort to supervise this dissertation. He has not only taught me the relevant materials in this field, but has also shown me crucial research skills that were necessary to complete this document. His insightful suggestions and knowledge in this field have shaped not only the way I view this topic, but my view of research as a whole. Without him, this dissertation would not have been possible.

I cannot fail to mention the love and support from my family. They have given me all that I could ever need and want. Many thanks goes to my Mum, Dad, my brother Reabetswe, uncles and aunts that have always wished the best for me. I would also like to mention Busi and Reggie Skhosana who have given me support and guidance for the past four years. Thank you for letting me be a part of your lives.

My special gratitude goes to the School of Computational & Applied Mathematics and National Research foundation (NRF) for their financial support towards my research.

A special thanks goes out to my friends and colleagues who have helped me at every of the way. Thanks to Dario for the lots of insightful discussions. Thank you to Tumelo for the advice and encouragement he gave me. Thank you to Kakanyo, Naval, Charles, Morgan, Viren, Asha, Franklin, Gideon, Terry, Tanya, Elimpoto, Innocent, Serge, Guo-Dong, Ricky and Carrie.

Last but not least, I thank my Bontle. I thank her for all the love and support she has given me. Her patients has had no boundaries. She has been an integral part of life for the past four years. This dissertation could not have happened without you.

Contents

1 Introduction **1**

1.1 Introduction 1

1.2 Problem formulation 2

1.3 Classification of global optimization algorithms 3

1.4 The structure of the dissertation 6

2 The multi-start technique and local search procedures **8**

2.1 Introduction 8

2.2 Discrete local search (DLS) 10

2.3 Model-based local search (MBLS) 11

2.3.1 MBLS for bound constrained problems 11

2.3.2 MBLS for constrained problems 13

2.3.3 A refined MBLS 15

2.4 Multi-start 16

2.5 Topographical clustering 18

2.6 Summary 20

3 Approximation algorithms for nonlinear integer programming (NIP) **21**

3.1 Introduction 21

3.2 Reformulation-based algorithms for NIP 22

3.2.1 Filled function methods 22

3.2.2 Auxiliary function methods 28

3.3 Darwin and Boltzmann mixed strategy 37

3.3.1 Nonlinear integer programming by Darwin and Boltzmann mixed strategy 37

4 Voronoi diagrams and the RRT algorithm 40

4.1 Voronoi diagrams 40

4.1.1 Computing Voronoi diagrams 41

4.2 Rapidly-exploring random trees (RRT) 43

4.2.1 Introduction 43

4.2.2 Problem formulation 43

4.2.3 Properties of the RRT 45

4.2.4 Analysis of RRT's 47

4.3 RRT-based optimizer 49

4.4 The RRTOptv1 algorithm 51

5 Numerical results 55

5.1 Results of MBLS 55

5.2 Results of the topographical clustering 57

5.3 Results of the RRT-Optimizer 58

5.4 Results of the RRTOptv1 algorithm 62

5.5 Summary 65

6 Conclusion 66

A Test problems 68

B Tables of paramters 78

C Table of RRTOptV1 using biasing towards 3 best points found 81

D Table of RRTOptV1 using biasing towards a combination of 1 local minimizer and 2 best points in the RRT tree 83

List of Figures

1.1 Simple taxonomy of optimization algorithms 4
1.2 The structure of the dissertation 7

2.1 Drawback of DLS 11
2.2 Topographical illustration 19

3.1 An example of the function $G(t)$ used in equation (3.2) 33
3.2 Effects of increasing the parameter k in $T(x, k)$ 34

4.1 The Voronoi diagram of a random set of points S in the plane. 41
4.2 An example of an RRT extension. 43
4.3 RRT algorithm at 3 different stages 46
4.4 The effect of biasing 52

List of Tables

5.1 Performance of MBLS on Problem 5, $n = 5$. 56

5.2 Comparison of methods 59

5.3 Performances and comparisons of algorithms on problems 60

5.4 Performances and comparisons of algorithms on problems (RRTOptv1) 64

B.1 Parameters for RRTOpt 79

B.2 Parameters for RRTOptv1 80

C.1 Performances and comparisons of algorithms on problems 82

D.1 Performances and comparisons of algorithms on problems 84

Chapter 1

Introduction

1.1 Introduction

The field of global optimization is well-known and widely researched into. The aim of global optimization is to minimize (or maximize) an objective function within a given space, with or without constraints. Global optimization can be divided into two main classes: continuous and discrete. Continuous optimization involves minimizing a function on the real space R^n, i.e. the optimizing variables in the given space are real. However, minimizing a function in a discrete optimization problem requires the optimizing variables to take integer values (i.e. in the set $Z^n \subset R^n$). The structure of these two classes may be linear or nonlinear. A structure is classified to be linear if both the objective function and the constraints are linear. If either the objective function or one of the given constraints is nonlinear, then the structure is termed nonlinear.

In a nonlinear optimization problem, multiple optimal solutions may exist. These solutions are termed *local* optima. Local optima are found using greedy/local search methods. The 'best' local optima is called the *global* optimum. Finding the global optimum can be a very difficult task. Generally, nonlinear optimization problems are considered to be NP-hard [2]. Apart from being an interesting and difficult mathematical problem, nonlinear optimization has many applications. Applications of nonlinear optimization can be found in various areas of scientific computing, engineering, management science and operations research. Many real-world applications include portfolio selection, capital budgeting, process engineering, production

planning, computer networks and flexible manufacturing systems [4, 10].

Any global optimization problem can be classified by the properties of the objective function and the constraints. Problems that do not contain constraints, and are bounded by upper and lower bounds, are considered as *unconstrained* problems. A problem that does contain constraints and is bounded below and above is said to be *constrained*. Solving constrained problems usually requires that a constrained problem be converted to an unconstrained one by means of a penalty function method [9].

The aim of this research is to design an algorithm that can solve both constrained and unconstrained nonlinear integer optimization problems. The algorithm should:

- work for a wide range of constrained and unconstrained problems of any dimension,
- not depend on the properties of the objective function and constraints,
- be easy to implement,
- be efficient and robust.

Finding an algorithm that satisfies all the criteria above can prove to be difficult. However some progress has been made in recent years to develop such an algorithm. We shall review these algorithms in the literature. In the following section, we present the formal problem for nonlinear integer programming.

1.2 Problem formulation

We want to attain the global optimum for any nonlinear integer programming problems.
A nonlinear integer programming (NIP) problem can be defined mathematically as follows:

$$(P) \quad \begin{cases} \min & f(x), \\ \text{s.t.} & g_i(x) \leq 0 \quad i \in K, \\ & h_j(x) = 0 \quad j \in J, \\ & x \in X \bigcap Z^n, \end{cases}$$

where K and J are finite index sets, Z^n is the set of integer points in R^n, and $f(x)$, $g_i(x)$ and $h_j(x)$ are all real valued. X is a polyhedral set i.e. $X = [a, b]^n$. Some functions in problem (P)

are nonlinear. We define the feasible set S by

$$S = \left\{ x \in X \bigcap Z^n | g_i(x) \leq 0, i \in K, h_j(x) = 0, j \in J \right\}. \tag{1.1}$$

Without loss of generality, we consider only the global minimization problem since the global maximum can be found in the same way by reversing the sign of the objective function $f(x)$, i.e.,

$$\max_S f(x) = \min_S (-f(x)). \tag{1.2}$$

An integer point $x_0 \in X \bigcap Z^n$ is called a discrete global minimizer of problem (P), if $f(x) \geq f(x_0)$, for all $x \in S$.

An integer point $x_0 \in S$ is called a discrete local minimizer of the problem (P), if $f(x) \geq f(x_0)$, for all $x \in N(x_0) \bigcap S$. [1]

1.3 Classification of global optimization algorithms

Global optimization algorithms can be classified into two basic classes: deterministic and probabilistic. Figure 1.1 shows a simple taxonomy of global optimization algorithms. Deterministic algorithms utilize the structure of the objective function $f(x)$, the constraints $g_i(x)$ and $h_j(x)$, and the feasible structure S. If the dimension of the problem is too high, deterministic algorithms cannot find the solution in a reasonable amount of time.

By contrast, probabilistic methods do not utilize any structure of (P) and the feasible set S. This makes probabilistic algorithms more applicable for finding global optima of general NIP problems. These algorithms are based on the Monte Carlo approach and were initially introduced about 55 years ago. Probabilistic methods rely on repeated random sampling to compute their results. For this very reason, this dissertation focuses on probabilistic algorithms, in particular multi-start (MS) combined with some heuristics [2]. We briefly present some deterministic and probabilistic algorithms.

[1] For any $x \in Z^n$, a set of integer points $N(x) \subseteq Z^n$ is called a neighbourhood of the integer point x, if $\{x, x + e_i, x - e_i, i = 1, 2, \cdots, n\} \subseteq N(x)$, where e_i is a Standard Basis Vector.

[2] Heuristics is a part of optimization that uses information currently gathered by the algorithm to help decide which candidate solution should be tested next or how the next individual can be produced.

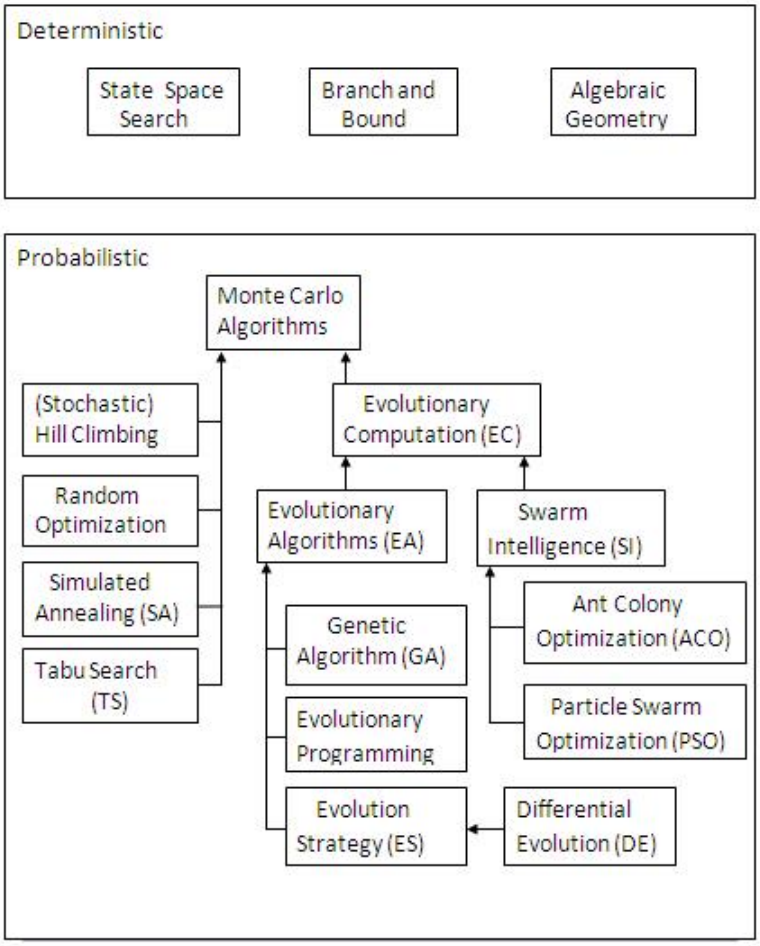


Figure 1.1: Simple taxonomy of optimization algorithms

The most commonly used deterministic algorithms is the branch and bound [1]. The idea of the branch-and-bound is to successively split (or branch) the problem into smaller problems, known as subproblems, which are easier to solve. Branching occurs at the nodes. A complete enumeration or solution of the subproblems that may arise during the branching process is not necessary. The branch-and-bound method has a special feature called fathoming, which consist of eliminating subtrees of the branch-and-bound tree from further consideration by pruning the corresponding root node. Another deterministic algorithm that is used is known as cutting plane. Cutting plane methods iteratively solve (P) by successively adding a linearization of the

most violated constraint at the predicted (current) solution[3]. If such a linearized constraint is found, then it is called a cut. The cut is added to the relaxation of (P). The cutting process is continued until an optimal solution is found. In another line of research, Srivastava and Fahim [7] presented a deterministic approximate method for solving (P). In the proposed optimization procedure, the minimization of $f(x)$ is carried out as usual along the function gradient, and a move back into the feasible region is achieved every time the constraints are violated. The search for the desired optimum, thus remains close to the constraint boundaries inside the feasible region. The results of the algorithm are, however, reported for problems of dimensions less than ten only [7].

Probabilistic algorithms can be classified into two groups: single-point based and population-based. Single-point based algorithms include tabu search [26], pure random search [24], multi-start [24], trajectory-based algorithms [4] and simulated annealing [8]. The multi-start type algorithms are also known as two-phase methods: global and local phase. In the global phase, the function is evaluated at a number of randomly sampled points (uniform distribution). In the local phase, the sample points are manipulated e.g., by means of local search, to yield a candidate global optimum.

Population based methods include differential evolution (DE) [5], particle swarm optimization (PSO) [3], ant colony optimization and genetic algorithm (GA) [23]. These methods start with many individual solutions which are randomly generated in the search space $X \bigcap Z^n$ to form an initial population. Then a proportion of the existing population is selected to breed a new generation by means of genetic operators: crossover and mutation or a combination thereof. Then, a better population set is obtained by replacing some (or all) the members of the initial population. The mechanism used to create the new population depends on the method being used. For example, in genetic algorithm, trail points (new generation) are generated by selecting successively a subset of the population and then applying mutation and crossover operations to the set. In differential evolution, trail points are also generated using crossover operations and mutation. In particle swarm optimization [3], individuals (called particles) change their position with time. Each particle adjusts its flying position according to its own experience, and according to flying experience of neighbouring particles, making use of the best position

[3]When (P) is a linear integer programming problem, a valid cutting plane is introduced at each iteration.

encountered by itself and its neighbours. This is how the new generation is formed.

1.4 The structure of the dissertation

The dissertation is divided into seven chapters as shown in Figure 1.2.

In Chapter two, we present a basic local (greedy-type) search algorithm that has been proposed for NIP [2]. We highlight the advantages and disadvantages of this local search algorithm. Moreover, we propose a local search of our own, called model-based local search (MBLS), which counters these disadvantages. We also present an overview of the topographical clustering method and show its usefulness in solving problem (P).

In Chapter three, we introduce approximate algorithms and reveal their strengths and weaknesses. We first review one of the most recent algorithms introduced by Zhu and Fan [2]. A lengthy and in-depth discussion is presented on it. The same is done for a variation of this algorithm proposed by Zhu and Ali [6].

In Chapter four, we give a brief introduction into Voronoi diagrams. We also give an in depth analysis on the rapidly-exploring random tree (RRT) method and highlight some of its key features. We propose the RRT-optimizer algorithm (RRTOpt) which combines RRT and topographical clustering in order to find the global optimum of (P). We also propose the probabilistic RRTOptv1 which is a modification of RRTOpt.

In Chapter five we test the new algorithms on some well known test problems, and compare them with recently developed algorithms. We also show that RRT-Opt is robust and more efficient than recently developed algorithms.

In Chapter six, we summarize the work done in this dissertation and propose further advances to extend and enhance the research. Finally, we present a collection of 20 benchmark nonlinear integer programming test problems in Appendix A.

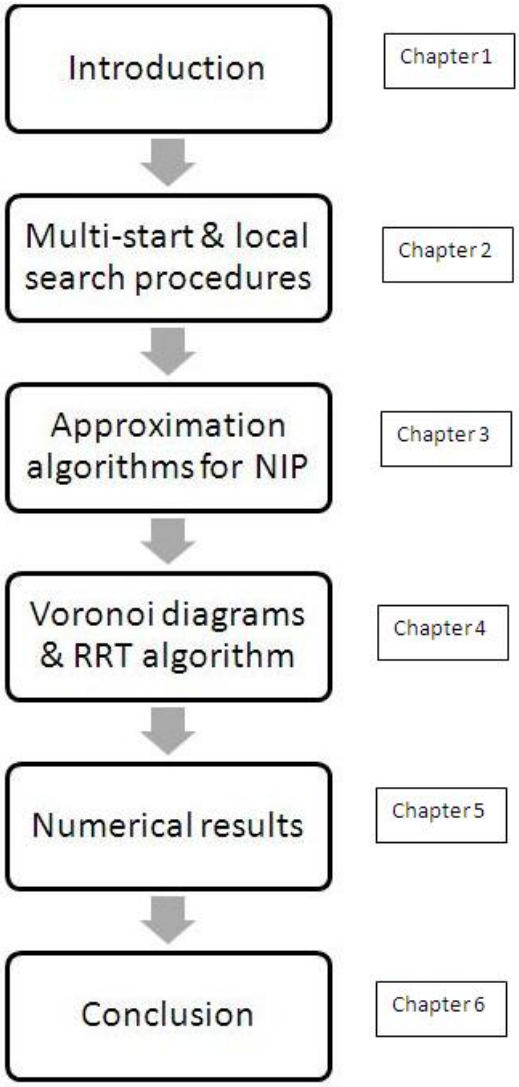


Figure 1.2: The structure of the dissertation

Chapter 2

The multi-start technique and local search procedures

In this chapter, we first discuss the local search procedure that has been used to find a local minimizer say, x_L, from a randomly selected point $x_0 \in X \bigcap Z^n$. Secondly, we will shift our focus to the Multi-start (MS) procedure for solving the global optimization problem. Since the most promising NIP algorithms appear to use the Multi-start technique as their building block, we thought it was worthwhile to discuss it in some depth. Lastly, we discuss a clustering method called topographical clustering which has some desirable properties in solving global optimization problems.

2.1 Introduction

In computer science, local search is a meta-heuristic[1] for solving computationally hard optimization problems. Local search algorithms are widely applied to numerous problems (other than computer science) from mathematics, operations research, engineering and bioinformatics. Local search algorithms move from solution to solution in the search space, X, of candidate solutions until a solution that is deemed optimal is found or a time bound is elapsed.

A local search algorithm starts from a candidate solution and then iteratively moves to a

[1]In the field of computer science, meta-heuristic designates a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality.

neighbour solution. This is only possible if a neighbourhood relation is defined on the search space X. Local optimization with neighbourhoods that involve changing up to k components of the solution is often referred to as *k-opt*.

Typically, every candidate solution has more than one neighbour solution; the choice of which one to move to is taken using only information about the solutions in the neighbouhood of the current one, hence the name *local* search. Given a candidate solution, neighbours are repeatedly generated until a better solution is found.

The generation of neighbours can be accomplished in different ways, namely [12]:

- *best improvement*, where all the neighbours are generated in lexicographic order and the best one, if better than the currrent solution, is chosen as the next solution.
- *first improvement with lexicographic generation*, where the neighbours are generated in lexicographic order and the first generated solution, that is better than the current one, is chosen as the next solution.
- *first improvement with random generation*, where the neighbours are generated randomly and the first generated solution, that is better than the current one, is chosen as the next solution.
- *Metropolis.* Given a solution s, its neigbours are explored randomly and a solution s' is always accepted if $f(s') < f(s)$, otherwise it is accepted with the probability $\exp(\frac{f(s)-f(s')}{t})$, where t is a parameter called temperature.

There are a number of ways to terminate a local search algorithm. The termination of a local search algorithm can be based on time bound. Another common choice is to terminate when the best solution found by the algorithm has not been improved in a given number of steps. Local search algorithms are typically incomplete algorithms, as the search may stop even if the best solution found by the algorithm is not optimal.

We now turn our focus to the *discrete* local search popularly used in nonlinear integer programming (NIP).

2.2 Discrete local search (DLS)

We begin this section with a brief discussion of the existing discrete local search techniques used in NIP [2, 25, 27, 6]. Thereafter we will follow up with a model-based local search that we have developed.

For any given initial starting point x_0, DLS creates a point $x_i = x_0 + d^i$ $(i = 1, 2, \cdots, 2n)$ in the neighbourhood of x_0, $d^i \in \left\{d^1, d^2, \cdots, d^{2n}\right\}$ where

$$\left\{d^1, d^2, \cdots, d^{2n}\right\} = \{e_1, e_2, \cdots, e_n, -e_1, \cdots, -e_n\}, \tag{2.1}$$

and e_i is a Standard Basis Vector. If $f(x_i) > f(x_0)$ then the next direction, d^{i+1}, is chosen to create $x_{i+1} = x_0 + d^{i+1}$ and the process is repeated for all d^i if $f(x_i + d^i) \geq f(x_0)$ for all i. In this case x_0 is treated as a discrete local minimizer. On the other hand, if $f(x_i + d^i) < f(x_0)$ for some direction d^i, then x_0 is updated as $x_0 = x_i + d^i$ and the process restarts again at x_0. The DLS algorithm is presented below.

The DLS algorithm [2, 27, 6]

Step 1. Take an initial integer point $x_0 \in X \bigcap Z^n$.

Step 2. If x_0 is a discrete local minimizer of $f(x)$ over $X \bigcap Z^n$, then stop; else take an integer point $x \in N(x_0) \bigcap X$ such that $f(x) \leq f(x_0)$.

Step 3. Let $x_0 := x$, and go to Step 2.

A drawback of DSL lies in the way it searches around the neighbourhood of the starting point x_0 in Step 2 in the above algorithm. We use the Figure 2.1 to illustrate the drawback. Suppose a better local minimizer lies along the direction $d^j = -e_i$ from x_0, that is along direction numbered three in Figure 2.1, and a worse local minimizer lies along $d^i = e_i$, along direction 7, where $i < j$. The worst local minimizer along the direction of $d^i = e_i$ will be found if $f(x_0 + d^i) < f(x_0)$. This could lead to the global minimizer, x^*, of (P) being missed by a particular iteration of the local search. Another variation of Step 2 is that the search direction can be random around $N(x_0)$, instead of co-ordinate-wise. To remedy the drawbacks of DLS, we propose a model-based local search and motivate its usefulness.

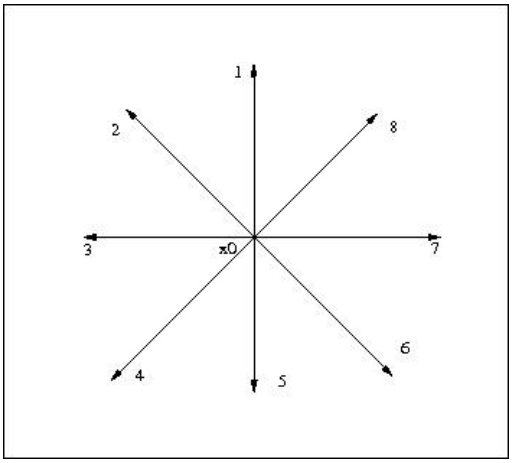


Figure 2.1: Drawback of DLS

2.3 Model-based local search (MBLS)

The model-based local search introduced here can be applied to (P) as well as to the bound constrained version of (P), i.e. minimize $f(x)$ subject to $x \in X \bigcap Z^n$. Implementation of MBLS to both versions are discussed. We begin with the implementation of MBLS to the bound constrained version of (P).

2.3.1 MBLS for bound constrained problems

MBLS approximates $f(x)$ by a quadratic function $q(x)$ in the vicinity of an initial starting point, say x_0. The steps of MBLS can be summarized as follows:

- Select a point x_0 to start the local search MBLS.
- Compute n neighbouring points, in the vicinity of x_0, and construct the quadratic $q(x)$ using n points around x_0.
- Use the information found by the approximated quadratic to select an appropriate search direction and compute a local minimizer of $f(x)$.

Given the point x_0, MBLS creates a set of n points

$$\{x_1, x_2, \cdots, x_n\} = \left\{x_0 + d^1, \cdots, x_0 + d^n\right\}, \tag{2.2}$$

where the directions d^i are not necessarily the same as in DLS (where unit coordinate vectors are used as direction vectors, see (2.1)). In MBLS, out of the n direction vectors, $n-h$ direction vectors are taken as the unit coordinate vectors. The remaining h direction vectors, e.g. $h=2$, are calculated by using any linear combination of the unit coordinate vectors. This serves for creating some flexibility in the local search. Each $d^i \in \left\{d^{n-h+1}, d^{n-h+2}, \cdots, d^n\right\}$ is given by

$$d^i = \sum_{j=1}^{r} D_j,\ d^i \neq 0, \tag{2.3}$$

where r is a random integer in $\{2, \cdots, n\}$ and $\{D_1, D_2, \cdots, D_r\}$ are distinct vectors in $\left\{d^1, d^2, \cdots, d^{2n}\right\}$ given in (2.3).

In the quadratic approximation, the linear and constant term are not considered. That is, we write $q(x) = \frac{1}{2}x^T Q x$, where Q is a $n \times n$ diagonal coefficient matrix whose elements are determined using the n points created close to x_0. We then obtain all descent[2] directions of $q(x)$ at x_0. Without loss of generality, assume that $\left\{\underline{d}^1, \underline{d}^2, \cdots, \underline{d}^p\right\}$ and $\left\{\bar{d}^1, \bar{d}^2, \cdots, \bar{d}^{n-p}\right\}$, respectively, are the set of descent and non-descent directions at x_0, obtained from the set of direction vectors $\left\{d^1, d^2, \cdots, d^n\right\}$.

During the construction process of $q(x)$, n neighbouring points, $\{x_1, x_2, \cdots, x_n\}$, of x_0 are calculated as in (2.2). Hence, for each $\underline{d}^j \in \left\{\underline{d}^1, \underline{d}^2, \cdots, \underline{d}^p\right\}$ there exists a corresponding $x \in \{x_1, x_2, \cdots, x_p\}$ such that $x = x_0 + \underline{d}^j$. Without the loss of generality, we denote the point by x_j, i.e. $x_j = x_0 + \underline{d}^j$. Hence $\underline{d}^j \in \left\{\underline{d}^1, \underline{d}^2, \cdots, \underline{d}^p\right\}$ corresponds to $x_j \in \{x_1, x_2, \cdots, x_p\}$. MBLS algorithm further investigates each $x_j \in \{x_1, x_2, \cdots, x_p\}$ along the direction $\underline{d}^j$. In particular, for each x_j, starting with $y_k = x_j$, $k = 0$, we calculate $y_{k+1} = y_k + \underline{d}^j$ and compare $f(y_k)$ with $f(y_{k+1})$. If $f(y_{k+1}) < f(y_k)$ then the next point y_{k+2} ($y_{k+2} = y_{k+1} + \underline{d}^j$) along $\underline{d}^j$ is obtained and the process is repeated until a point y_{k+l}, along $\underline{d}^j$, is found for which $f(y_{k+l}) \geq f(y_{k+l-1})$. Then we set $\overline{y}_j = y_{k+l-1}$ and continue the procedure for all remaining $x_j \in \{x_1, x_2, \cdots, x_p\}$. Notice that if $f(y_{k+1}) > f(y_k)$, $f(y_k) = f(x_j)$, then $\overline{y}_j = x_j$.

Once the procedure is complete then we stop the MBLS algorithm and output the point

$$x_L = \arg\min\left\{f(\overline{y}_1), f(\overline{y}_2), \cdots, f(\overline{y}_p), f(x_0), f(x_1), \cdots, f(x_{n-p})\right\} \tag{2.4}$$

as the local minimizer. In (2.4), $\{x_1, x_2, \cdots, x_{n-p}\}$ is the set of points corresponding to the set of non-descent directions $\left\{\bar{d}^1, \bar{d}^2, \cdots, \bar{d}^{n-p}\right\}$, i.e. $x_j = x_0 + \bar{d}^j$. We now present the MBLS

[2] A direction d is descent at some point x_0 if $d^T \nabla q(x_0) < 0$.

algorithm.

The MBLS algorithm

Step 1. Take an initial integer point $x_0 \in X \bigcap Z^n$.

Step 2. Compute a set of n integer points $\{x_i, i = 1, 2, \cdots, n\}$ using (2.2), and determine the coefficients of $q(x)$.

Step 3. Calculate the steepest descent directions, from x_0, using $d^{(i)T}\nabla q(x_0)$, $\forall i$. Let $R = \left\{j \in \{1, 2, \cdots, n\} \,|d^{(j)T}\nabla q(x_0) < 0\right\}$ be the set of indices corresponding to all descent directions from the starting point x_0. Denote the corresponding set of descent directions by $\left\{\underline{d}^1, \underline{d}^2, \cdots, \underline{d}^p\right\}$.

3.1 If $R = \phi$ (where ϕ denotes the empty set) go to Step 5, otherwise set $j = 1$ and go to Step 4.

Step 4. For $j \in R$ (or $\underline{d}^j$) set $k = 0$, $y_k = x_j$, and calculate $y_{k+1} = y_k + \underline{d}^j$.

4.1 If $f(y_{k+1}) < f(y_k)$, go to Step 4.2. Otherwise go to Step 4.3.

4.2 Update $y_k = y_{k+1}$, set $k = k + 1$, compute $y_{k+1} = y_k + \underline{d}^j$ and go to Step 4.1.

4.3 Let $\overline{y}_j = y_k$ and go to Step 4.4.

4.4 Increase j by 1, i.e. $j = j + 1$. If $j \leq p$ then go to Step 4, else go to Step 6.

Step 5. Set $x_L = \arg\min\{f(x_0), f(x_1), \cdots, f(x_n)\}$ and stop the algorithm.

Step 6. Set $x_L = \arg\min\left\{f(\overline{y}_1), f(\overline{y}_2), \cdots, f(\overline{y}_p), f(x_0), f(x_1), \cdots, f(x_{n-p})\right\}$ and stop the algorithm.

Remark 1. *A property of MBLS is that when $q(x)$ closely approximates the function $f(x)$, the descent directions of $q(x)$ at x_0 will be the descent directions of $f(x)$ at x_0.*

2.3.2 MBLS for constrained problems

When solving the problem (P), we construct the *penalty function* [9]:

$$F(x; c) = f(x) + c\phi(x), \tag{2.5}$$

where c is a *penalty parameter*, and

$$\phi(x) = \sum_{i \in K} \max\{0, g_i(x), i \in K\} + \sum_{j \in J} |h_j(x)| \tag{2.6}$$

measures the constraint violations. MBLS for constrained (P) works exactly the same way it works for the unconstrained problem (P). However, due to the presence of constrained violations, $\phi(x)$, some adjustments are made. In particular, the quadratic $q(x)$ is found using the values of $F(x;c)$ instead of $f(x)$ as in the unconstrained problem.

As in Section 2.3.1, we have the set of points

$$\{x_1, x_2, \cdots, x_p\} \tag{2.7}$$

close to x_0 along the set of directions $\left\{\underline{d}^1, \underline{d}^2, \cdots, \underline{d}^p\right\}$. For each $\underline{d}^j \in \left\{\underline{d}^1, \underline{d}^2, \cdots, \underline{d}^p\right\}$, we set $y_k = x_j$ and two comparisons are made. In particular, $F(y_k;c)$ is compared with $F(y_{k+1};c)$, and $\phi(y_k)$ is compared with $\phi(y_{k+1})$, where the points y_k and y_{k+1} are such that $y_k = x_j$ and $y_{k+1} = y_k + \underline{d}^j$. From this comparison, four cases arise.

Case 1:

If $F(y_{k+1};c) > F(y_k;c)$ and $\phi(y_{k+1}) \geq \phi(y_k)$ then let $\overline{y}_j = y_k$ and end the search along $\underline{d}^j$.

Case 2:

If $F(y_{k+1};c) < F(y_k;c)$ and $\phi(y_{k+1}) \geq \phi(y_k)$ then

(i) If $\phi(y_{k+1}) \neq 0$ then let $\overline{y}_j = y_k$ and end the search along $\underline{d}^j$.

(ii) If $\phi(y_{k+1}) = 0$ then increase k by one i.e. $k = k + 1$ (calculate the next point along $\underline{d}^j$) and consider the appropriate case amongst 1,2,3 and 4 (between two consecutive points calculated).

Case 3:

If $F(y_{k+1};c) > F(y_k;c)$ and $\phi(y_{k+1}) \leq \phi(y_k)$ then

(i) If $\phi(y_{k+1}) \neq 0$ then let $\bar{y}_j = y_{k+1}$ and end the search along $\underline{d}^j$.

(ii) If $\phi(y_{k+1}) = 0$ then let $\bar{y}_j = y_k$ and end the search along $\underline{d}^j$.

Case 4:

If $F(y_{k+1};c) \leq F(y_k;c)$ and $\phi(y_{k+1}) < \phi(y_k)$ then increase k by one i.e. $k = k+1$ (calculate the next point along $\underline{d}^j$) and consider the appropriate case amongst 1,2,3 and 4 (between two consecutive points calculated).

After MBLS has stopped, we identify the number of feasible points in the set $\{\bar{y}_1, \bar{y}_2, \cdots, \bar{y}_p, x_0, x_1, \cdots, x_{n-p}\}$ and treat the best feasible point as the local minimizer, x_L. If there are no feasible points in $\{\bar{y}_1, \bar{y}_2, \cdots, \bar{y}_p, x_0, x_1, \cdots, x_{n-p}\}$ then x_L is found by

$$x_L = \arg\min \{F(\bar{y}_1;c), F(\bar{y}_2;c), \cdots, F(\bar{y}_p;c), F(x_0;c), F(x_1;c), \cdots, F(x_{n-p};c)\}. \tag{2.8}$$

A refined version of the MBLS algorithm is presented in the following subsection.

Remark 2. *Various cases are considered in MBLS to counter the wrong choice of the penalty parameter c (the optimal choice of c is a difficult issue).*

2.3.3 A refined MBLS

In the refined version of MBLS (denoted MBLSr), we consider the *quadratic approximation* function $q(x) = \frac{1}{2}x^TQx$ and calculate the $n \times n$ symmetric coefficient matrix Q. Since Q is symmetric, it has $\frac{n(n+1)}{2}$ distinct entries. Therefore $\frac{n(n+1)}{2}$ number of points, around x_0, are required to calculate the matrix Q. Thus we use the set

$$\left\{d^1, d^2, \cdots, d^l\right\}, l = \frac{n(n+1)}{2}, \tag{2.9}$$

as the set of direction vectors to create the set of points $\{x_1, x_2, \cdots, x_l\}$, using (2.2), from a starting point x_0.

During the construction of the set in (2.9), two cases are considered. If $n \leq 3$ then the subset of direction vectors $\left\{d^1, d^2, \cdots, d^{l-h}\right\}$ comprises of unit coordinate vectors (for some integer $h < l$), and the remaining subset of direction vectors $\left\{d^{l-h+1}, \cdots, d^l\right\}$ is constructed using (2.3), section 2.3.1. When $n > 3$, however, the subset of direction vectors $\left\{d^1, d^2, \cdots, d^{2n}\right\}$ comprises of unit coordinate vectors only. The remaining subset of direction vectors $\left\{d^{2n+1}, d^{2n+2}, \cdots, d^l\right\}$ is constructed using (2.3), subsection 2.3.1.

Motivation for using MBLS is shown in a later section. We now turn our attention to the multi-start algorithm which has a local search procedure embedded in it.

2.4 Multi-start

Before our discussion of the multi-start technique [24] begins , we first briefly discuss the most basic stochastic method for global optimization known as *Pure Random Search*. Pure random search involves no more than a single step. Despite being a very simple method, it offers an asymptotic guarantee in a probabilistic sense. The method iteratively samples points in the sample space, $X \cap Z^n$, and evaluates them until a stopping criterion is satisfied. Then, the algorithm approximates the point with the lowest function value found as the global minimizer of the problem.

However, pure random search is not taken seriously as a computational proposal. Several extensions of this method have been proposed that also start from a uniform sample over X, but that at the same time involve local searches from some or all points in the sample. One of these extensions is indeed the multi-start technique.

Multi-start is a technique where points are sampled iteratively from a uniform distribution over X (global phase), after which local minima will be found by applying a local search procedure to these points (local phase). Thereafter, if some stopping rule is met, the multi-start algorithm is terminated. We state the formal multi-start algorithm:

Multi-start [24]

Step 1. Draw a sample point, x_0, from a uniform distribution over the search space X.

Step 2. Apply a local search technique to the sample point.

Step 3. A termination criterion indicates whether to stop or to return to Step 1. The local minimizer with the smallest function value found is the candidate for the global minimizer.

The stopping rule termination criterion in Step 3 of the above algorithm is based on a *Bayesian* estimate of the number of local minima W_i and the relative size, Θ , of each region of attraction[3], R_{x^*}, of the local minimum, x^*, say. The relative size of the region of attraction is given by $\Theta_i = \frac{m(R_{x^*})}{m(X)}$, where $i = 1, \cdots, W$ and $m(\cdot)$ is the Lebesgue measure. If the values of these parameters were given, then it would be possible to determine the possibility that W different local minima are found during the N local searches. This probability can be used in a Bayesian

[3]This is the set of points in X starting from which a local search technique will converge to x^*

approach in which the unknowns W, $\Theta_1, \Theta_2, \cdots, \Theta_W$ are assumed to be themselves random variables for which a *prior distribution* can be specified. Given the outcome of an application of the multi-start algorithm, Bayes' rule is used to compute the *posterior* distribution[4], which incorporates both the prior beliefs and the sample information. Simple expressions emerge for the posterior distribution and posterior expectation of several interesting parameters, some of which are stated in the following Theorem.

Proposition 1. *[24] If w different local minima have been found as the result of N local searches started in uniformly distributed points, if we assume a prior for a number of local minima $\underline{W}$ that each integer of $[1, \infty)$ is equally probable, and if we assume that the given $\underline{W} = W$ the relative sizes of the regions of attraction $\Theta_1, \Theta_2, \cdots, \Theta_W$ following a uniform distribution on the $(W-1)$-dimension unit simplex, then*

(i) the posterior probability that there are K local minima is equal to

$$\frac{(K-1)!K!(N-1)!(N-2)!}{(N+K-1)!(K-w)!w!(w-1)!(N-w-2)!} \tag{2.10}$$

(ii) the posterior expectation of the number of local minima is

$$\frac{w(N-1)}{N-w-2} \tag{2.11}$$

provided that $N > w+2$

(iii) the posterior expected relative size of the non-observed regions of attraction is

$$\frac{w(w+1)}{N(N-1)}. \tag{2.12}$$

This theoretical framework proposed by the authors is quite an attractive one, the more so since it can be easily extended to yield *optimal Bayesian stopping rules*. Such rules incorporate an assumption about the cost and potential benefits of further experiments and weigh these against each other probabilistically to calculate the optimal stopping point. Two observations conclude this Bayesian analysis. First, note that the posterior distributions and expectations do not depend on the dimension of the problem. The number of local searches that has to be

[4] In Bayesian statistics, the posterior probability of a random event or an uncertain proposition is the conditional probability that is assigned after the relevant evidence is taken into account.

performed only depends on the number of minima located. Secondly, the a priori assumption that every number of local minima is equally probable may appear to be very pessimistic.

Although multi-start is obviously better than pure random search, there are several inefficiencies that remain. The main inefficiency is that multi-start will inevitably cause each local minimizer to be found several times. To avoid all these time consuming local searches, the local search should ideally be invoked no more than once in every region of attraction.

There have been some attempts to modify the multi-start technique. One such attempt involved invoking a local search to the point that is drawn (in Step 1) that has a smaller function evaluation than the smallest minimum that has been found. It should be obvious that under this rule, the global minimum may not be found even if a point is sampled in R_{x^*}. Another adaptation of the Multi-start has however been more successful and is provided by *clustering methods* [24]. One such clustering method was presented by Törn [20].

2.5 Topographical clustering

Topographical clustering [20] is a clustering technique whereby the centres of each cluster are identified. These centres are known as the *graph minimizers*. The concept was suggested in the context of global optimization involving continuous variable [20]. Given a number of points, say N, generated uniformly in the search space, graph minimizers are calculated to perform local search from them. Topographical clustering has never been applied to integer programming. We explain the concept with the aid of a simple one-dimensional function $f(x)$, $x \in [a, b]$.

Suppose the function $f(x)$ in Figure 2.2 has three local minimizers in the interval $[a, b]$. In order to obtain the graph minimizers, we randomly select N points ($N = 10$) in $[a, b]$. These points are numbered in Figure 2.2 and are denoted by the set $\{x_i = i, i = 1, 2, \cdots, N\}$. Their corresponding $f(x_i)$ are presented in the y-axis. Central to the calculation of the graph minimizers is the number g_m of nearest neighbouring points of each x_i, $i = 1, 2, \cdots, N$. For each point, x_i, we consider a number of nearest neighbours, say $g_m = 2$. We now study each point and identify the points, x_i, for which each of its g_m ($g_m << N$) nearest neighbour has a higher function value than $f(x_i)$. This may be done by calculating a distance matrix D for the N points as well as

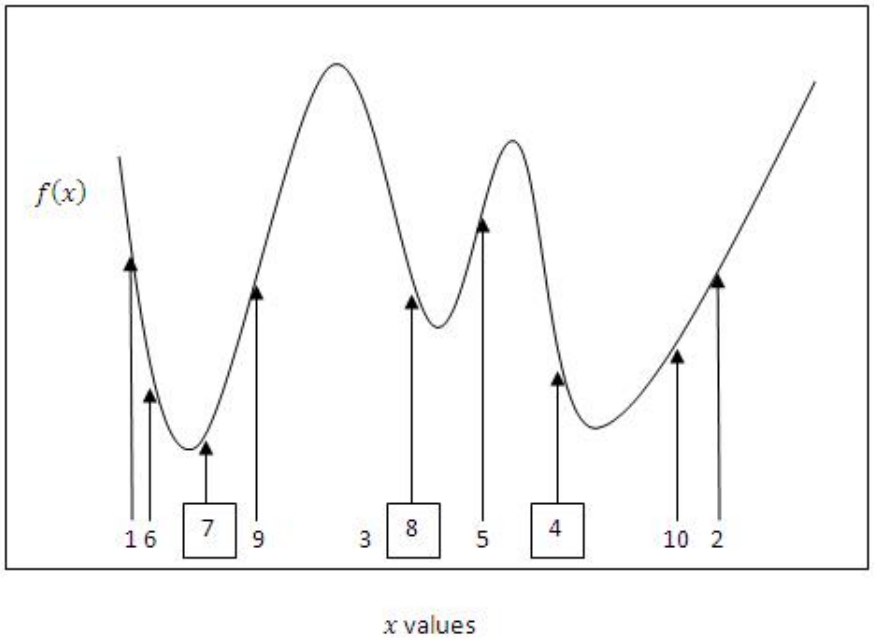


Figure 2.2: Topographical illustration

checking whether the g_m nearest neigbouring points have a higher function value for each x_i. For example, if we take $x_6 = 6$, its two nearest neighbours are $x_1 = 1$ and $x_7 = 7$, but $f(x_7) < f(x_6)$. Hence, x_6 is not a graph minimizer. If however, we take $x_7 = 7$, its two nearest neighbours are x_6 and x_9 but $f(x_7) < f(x_9)$ and $f(x_7) < f(x_6)$. Hence x_7 is a graph minimizer. Applying this procedure, we can see that graph minimizers are at x_8, x_7 and x_4. We have presented these graph minimizers in Figure 2.2 within the boxes. These points are separated from each other and they are also close to the minimizers of $f(x)$ in $[a, b]$. For the constrained (P), the penalty function $F(x; c)$ is used in obtaining the graph minimizers.

The multi-start algorithm proposed in this section is then used in conjunction with a local search routine, but this local search routine is applied to the graph minimizers only. Since, the graph minimizers are points with lower function values as well as they are separated from each other, applying local search from these points has the following advantages. Firstly, the local minimizers are detected with less efforts and secondly, the probability of a repetition of the same

local minimizers will be less.

2.6 Summary

In this chapter, we have discussed local search techniques and the multi-start method in some depth. We have proposed our own local search procedure (MBLS) that counters the inefficiencies of the commonly used DLS. The topographical clustering has been reviewed. Local search is only invoked on the graph minimizers, which seems to be more efficient than applying local search on any random point found. We now shift our focus to approximate stochastic algorithms that attempt to solve problem (P) using the multi-start and DLS as their foundation.

Chapter 3

Approximation algorithms for nonlinear integer programming (NIP)

Approximation algorithms have been used to solve general nonlinear integer programming (NIP) problems with some degree of success [2, 6]. The difficulty for searching for a global optimum value presents two sides: how to leave from a local minimizer to a better local minimizer (one whose function evaluation is lesser than the previous one) and how to judge if the current minimizer is a global one [22]. In this chapter, we describe the approximation algorithms that have been used to solve problem (P) and highlight some of their advantages and disadvantages.

3.1 Introduction

In computer science and operations research, approximation algorithms are algorithms that are used to find approximate solutions to global optimization problems. Approximation algorithms are often associated with NP-hard problems [27] (since it is very unlikely that there can ever be an efficient polynomial time exact algorithm for solving NP-hard problems). So approximate algorithms or heuristic methods are particularly important for nonlinear integer programming problems, especially for high-dimensional cases [2]. The goal of approximation algorithms is

to find the optimum solution, or get to it as close as possible, in a reasonable amount of time (which is at most polynomial time).

All *exact* algorithms that aim to solve (P) have *computational complexity* [27]. Because of this reason, approximation algorithms have been developed to solve the different classes of NIP problems. Approximate algorithms or heuristic methods developed for nonlinear integer programming are very limited, and often have few computational experiments [2]. These algorithms can be classified into two classes: probabilistic (stochastic) and deterministic. These two classes were mentioned in the introductory chapter of this dissertation. Next we present some probabilistic approximation algorithms that have shown some limited success in solving (P).

3.2 Reformulation-based algorithms for NIP

Reformulation-based algorithms minimize *auxiliary* and *filled* functions instead of $f(x)$ in problem (P) [27, 2, 6, 25]. This clearly implies that the filled and auxiliary functions have the same global optimum as $f(x)$. We present some algorithms that use the filled functions in order to solve (P).

3.2.1 Filled function methods

A filled function is a function that has the same discrete local minimizer, x^*, as $f(x)$ in the region near x^*. The first filled function method we will outline was created by Zhu [27]. This method is the discrete case of the method presented by Ge [22]. The method tries to improve a current local minimum solution by minimizing a filled function. Suppose that a discrete local minimizer x_1^* has been found by using discrete local search (DLS). In order to find a global optimum solution, another discrete local minimizer is found, x_2^*, such that $f(x_2^*) < f(x_1^*)$, which can be fulfilled by minimizing a filled function. We now see how Zhu defined the filled function he used in this particular method where $D = \{e_i, -e_i, i = 1, 2, \cdots, n\}$.

Definition 1. *[27] An integer point $x_0 \in X \cap Z^n$ is a discrete local maximal solution to problem (P) if $f(x_0 + d) \leq f(x_0)$, for all $d \in D$ such that $x_0 + d \in X \in Z^n$. The filled function is constructed by:*

$$P(x, r, \rho) = \frac{1}{r + f(x)} \exp(-\frac{\|x - x_1^*\|^2}{\rho^2}) \tag{3.1}$$

where $\rho \neq 0$, r are parameters to be adjusted, $\|\cdot\|$ denotes the Euclidean norm.

We state the following theorems and lemma without any loss of generality where h is a parameter such that $h > 0$. The proofs can be found in [27]. These theorems and lemma will be used later in the explanation of Zhu's algorithm.

Theorem 1. *Let r, ρ satisfy $0 < r + f(x_1^*) < h$ and $\rho^2 \ln \frac{r+\overline{f}}{r+f(x_1^*)} < 1$.*
For $x_1 \in X \cap Z^n$, $x_1 \neq x_1^*$, and $f(x_1) \geq f(x_1^*)$,

1. *if there exist $d \in D$ such that $x_1 + d \in X \cap Z^n$ and $P(x_1 + d, r, \rho) < 0$ or*
2. *if $|\{d \in D : x_1 + d \in X \cap Z^n\}| = n$, and there exists $d \in \{d \in D : x_1 + d \in X \cap Z^n\}$ such that $P(x_1 + d, r, \rho) < P(x_1, r, \rho)$ or*
3. *if $|\{d \in D : x_1 + d \in X \cap Z^n\}| > n$,*

then there exists some $d \in D$, such that $x_1 + d \in X \cap Z^n$ and $P(x_1 + d, r, \rho) < P(x_1, r, \rho) < P(x_1^*, r, \rho)$.

Theorem 2. *Let r, ρ satisfy the conditions of Theorem 1. Then any one of the discrete local minimal solutions of the filled function $P(x, r, \rho)$ over $X \cap Z^n$ must be in the set $S = \{x \in X \cap Z^n : P(x, r, \rho) < 0\}$ or a vertex of the bounded box X.*

Lemma 3. *Suppose that $0 < r + f(x_1^*) < h$. $x \in X \cap Z^n$. For any $d \in D$, such that $x_1 + d \in X \cap Z^n$, we have:*

1. *$P(x_1 + d, r, \rho) < 0 \Leftrightarrow f(x_1 + d) < f(x_1^*)$.*
2. *$P(x_1 + d, r, \rho) > 0 \Leftrightarrow f(x_1 + d) \geq f(x_1^*)$.*

We next present the formal algorithm, which is called the Approximate Algorithm, introduced by Zhu to find the global optimum of problem (P).

Algorithm 1: The approximate algorithm [27]

Step 0 Take an initial point $x_0 \in X \cap Z^n$.

Step 1 From x_0, start DLS to minimize $f(x)$ over $X \cap Z^n$. Denote by x_1^* a found discrete local minimal solution.

Step 2 Let $D = \{e_i, -e_i, i = 1 \cdots n\}$.

Step 3 Construct a filled function

$$P(x, r, \rho) = \frac{1}{r + f(x)} \exp\left(-\frac{\|x - x_1^*\|^2}{\rho^2}\right)$$

where

$$0 < r + f(x_1^*) < h, \quad \rho^2 \ln \frac{r + \bar{f}}{r + f(x_1^*)} < 1.$$

Let $x_1 = x_1^*$.

Step 4 From x_1, use DLS to minimize $P(x, r, \rho)$ over $X \cap Z^n$. Denote by x_2^* a found discrete local minimal solution.

Step 5 If $P(x_2^*, r, \rho) < 0$, let $x_0 := x_2^*$, go to Step 1.

Step 6 If $D = \emptyset$, go to Step 7; otherwise choose an element $d \in D$, let $D = D \setminus \{d\}$. If $x_1 + d \in X \cap Z^n$, $x_1 + d \neq x_1^*$, then let $x_1 = x_1 + d$ and go to Step 4; else repeat Step 6.

Step 7 Stop, x_1^* is an approximate discrete global minimal solution to problem (P).

We explain the above algorithm. Taking an initial point $x_0 \in X \in Z^n$, from which DLS is started to find a discrete local minimal solution of problem (P), say x_1^*. Then a filled function $P(x, r, t)$ is constructed, where r, p satisfy the conditions of Theorem 1. Thus taking x_1^* as an initial point, the approximate algorithm is initialized to minimize $P(x, r, t)$ over $X \cap Z^n$. If a discrete local minimal solution of $P(x, r, t)$, say x_2^*, is found such that $P(x, r, t) < 0$. i.e. $f(x_2^*) < f(x_1^*)$, by Lemma 3, x_2^* is used as a new initial point and go to the next iteration of the algorithm; otherwise by Theorem 2, x_2^* is a vertex of X and $P(x, r, t) > 0$. This kind of x_2^* is not needed, another initial point must be taken, instead of x_1^*, to minimize $P(x, r, t)$ again.

In Step 1 of the above algorithm, if x_1^* is a discrete global optimal solution to the problem (P), then by Theorem 2, x_2^* found in Step 4 must be a vertex of the bounded box X. Further more, in Step 3 of the algorithm, parameters r, p must be determined at first. If $f(x)$ is a polynomial function with integer coefficients, then $h = 1$ is set and parameters r, p are easy to determine. If the value of parameter h is not known at hand, then h is set to be sufficiently small, then r, p are also determined.

When testing the algorithm above, the authors set parameters p and r as:

$$r = \frac{h}{2} - f(x_1^*)$$

and

$$\frac{1}{p^2} = 1 + \ln\left(\frac{\bar{f} - f(x_1^*) + \frac{h}{2}}{\frac{h}{2}}\right)$$

where h, $\bar{f}$ are determined according to different problems.

This method for solving problem (P) does not seem to be very general. Firstly, the author tested his algorithm on two problems that were unconstrained. Secondly, the dimensions of these two problems are less than 15[1]. Some test problems of the form (P) are of dimension up to two hundred. Lastly, the author had to prescribe initial points to initialize his algorithm. This can lead to a biased analysis on the performance of the method . We now outline a method that seems to show better results than the Approximate Algorithm.

Wang et.al. [25] proposed a stochastic approximate algorithm which uses a filled function $P(x, x^*)$ to solve discrete general bound constrained minimization problems, i.e. $g(x)$ and $h(x)$, in (P), are zero. The function is called a T-F function, it is both filled and tunnel. The definition of a filled function and a tunnel function are as follows:

Definition 2. *$P(x, x^*)$ is called a discrete filled function of $f(x)$ at a discrete local minimizer x^* if $P(x, x^*)$ has the following properties:*

1. *x^* is a strict discrete local maximizer of $P(x, x^*)$ over $X \cap Z^n$.*

2. *$P(x, x^*)$ has no discrete local minimizers on the set*
   $\{x | f(x) \geq f(x^*), x \in A \backslash \{x^*\}\}$, where $A = X \cap Z^n$.

[1]The author tested the algorithm on two problems which are of dimensions four and ten. The initial points were prescribed and set to be equal to the zero vector for each problem.

3. *If x^* is not a discrete global minimizer of $f(x)$, then $P(x, x^*)$ does not have a discrete minimizer in the set $\{x|f(x) < f(x^*), x \in A\}$.*

Definition 3. *$P(x, x^*)$ is called a discrete tunnel function of $f(x)$ at a discrete local minimizer x^* if, for any $x^0 \in A$ with a parameter $r > 0$, $P(x, x^*) = 0$ if and only if $f(x^0) - f(x^*) + r \leq 0$.*

Wang and et.al.[25] define a function $T(x, x^*, r, q)$ of f at a given point $x^* \in X$ with $r > 0$ and $q > 0$ as follows

$$T(x, x^*, r, q) = \frac{1}{1 + \|x - x^*\|} h_r(f(x) - f(x^*) + r + 1) + q \max(0, f(x) - f(x^*))$$

where $0 < r < \min_{\substack{f(x_1) \neq f(x_2) \\ x_1, x_2 \in X}} |f(x_1) - f(x_2)|$. The function $h_r(t)$ is given by

$$h_r(t) = \begin{cases} 1, & t \geq 1 + r \\ \phi_r(t), & 1 \leq t \leq 1 + r \\ 0, & t \leq 1 \end{cases}$$

where $\phi_r(t)$ satisfies the following conditions:

For $1 < t < 1+r$, $1 \geq \phi_r(t) \geq 0$, $\phi_r(1) = 0$, $\phi_r(r+1) = 1$. It is easy to conclude that $T(x, x^*, r, q)$ is a tunnel function and Wang et.al. proved that it is also a filled function, thus making it a T-F function.

The algorithm developed by Wang and et.al. [25] constructs a T-F function at a local minimizer of the objective function such that it achieves local *maximum* at the current solution. A local minimizer of the T-F function leads to a new solution of the original problem with a lower function value. Iterations follow in this manner to reach a global minimizer. The discrete local search (DLS) technique is used in this algorithm to find local minimizers. We present the formal algorithm for this method called the Discrete T-F function method.

Algorithm 2 : Discrete T-F function method [25]

Step 1. Input the lower bound of r, namely $r_L = 10^{-8}$. Input an initial point $x_0^{(0)} \in X$. Let $D = \{\pm e_i, i = 1, 2, \cdots, n\}$.

Step 2. Starting from an initial point $x_0^{(0)} \in X$, minimize $f(x)$ and obtain the first local minimizer x_0^* of $f(x)$. Set $k := 0, q = 1$ and $r = 1$.

Step 3. Step $x_k^{(0)i} = x_k^* + d_i$, $d_i \in D, i = 1, 2, \cdots, 2n$, $J = [1, 2, \cdots, 2n]$ and $j = 1$.

Step 4. Set $i = J_j$ and $x = x_k^{(0)i}$.

Step 5. If $f(x) < f(x_k^*)$, then use x as an initial point for discrete local minimization method to find x_{k+1}^* such that $f(x_{k+1}^*) < f(x_k^*)$. Set $k = k + 1$, go to Step 3.

Step 6. Let $D_0 = \{d \in D \colon x + d \in X\}$. If there exists $d \in D_0$ such that $f(x + d) < f(x_k^*)$, then use $x + d^*$, where $d^* = \arg\min_{d \in D_0} \{f(x + d)\}$, as an initial point for a discrete local minimization method to find x_{k+1}^* such that $f(x_{k+1}^*) < f(x_k^*)$. Set $k \colon = k + 1$ and go to Step 3.

Step 7. Let $D_1 = \{d \in D_0 \colon \|x + d - x^*\| > \|x - x^*\|\}$. If $D_1 = \emptyset$ then go to Step 10.

Step 8. If there exists $d \in D_1$ such that $T(x + d, x_k^*, r, q) \geq T(x, x_k^*, r, q)$, then set $q = 0.1q$, $J = [J_j, \cdots, J_{2n}, J_1, \cdots, J_{j-1}]$, $j = 1$ go to Step 4.

Step 9. Let $D_2 \colon = \{d \in D_1 \colon f(x + d) < f(x), T(x + d, x_k^*, r, q) < T(x, x_k^*, r, q)\}$. If $D_2 \neq \emptyset$, then set $d^* \colon = \arg\min_{d \in D_2} \{f(x + d) + T(x + d, x_k^*, r, q)\}$; Otherwise set
$d^* \colon = \arg\min_{d \in D_1} \{T(x + d, x_k^*, r, q)\}$. After that set $x = x + d^*$ and go to Step 6.

Step 10. If $i < 2n$, then set $i = i + 1$ and go to Step 4.

Step 11. Set $r = 0.1r$. If $r \geq r_L$, go to Step 3; otherwise, the algorithm is incapable of finding a better minimizer starting from the initial points $\left\{x_k^{(0)i} \colon i = 1, 2, \cdots, 2n\right\}$. The algorithm stops and x_k^* is taken as a global minimizer.

We give a brief explanation of the above mentioned algorithm. A random point, $x_0^{(0)}$, and the stopping parameter, r_L, is inputted in step one. From this point DLS (with best improvement) is invoked, using the objective function $f(x)$, to obtain a local minimizer x_0^*. The parameters k,q and r are set. Steps 3,4,5 and 6 attempt to find the k-th discrete local minimizer for the i-th iteration (in the i-th direction). From Step 8, we see the introduction of the T-F function. If there exists a point around x such that $T(x + d, x_k^*, r, q) \geq T(x, x_k^*, r, q)$ then the parameter q is reduced, then $i = i + 1$ and Step 4 is invoked. Otherwise, if no point around x satisfies the criterion $f(x + d) < f(x)$ and $T(x + d, x_k^*, r, q) < T(x, x_k^*, r, q)$, the direction d^*, from x,

which minimizes $f(x) + T(x + d, x_k^*, r, q)$ is chosen and $x = x + d^*$ is set and the algorithm goes back to Step 6. If however there exists a point(s) around x such that $f(x + d) < f(x)$ and $T(x + d, x_k^*, r, q) < T(x, x_k^*, r, q)$ then, take direction d^* that minimizes $T(x, x_k^*, r, q)$ then set $x = x + d^*$ and the algorithm goes to Step 6. If D_1 is the empty set in Step 7 then the algorithm has reached the boundary point, hence the algorithm goes to Step 10 and the iteration is changed to $i = i + 1$ and the algorithm goes to Step 4 if $i < 2n$, otherwise the algorithm goes to Step 11 where the stopping parameter, r, is reduced. If the stopping criterion is met then the algorithm is incapable of finding a better discrete local minimizer from the initial points $\left\{x_k^{(0)i} : i = 1, 2, \cdots, 2n\right\}$. The algorithm is terminated and x_k^* is taken as the global minimizer of problem (P).

We were not able to compare the method of Zhu [27] and the method presented in [25] since a complementary set of problems were used. The one distinct advantage of Discrete T-F function method is that the initial point is randomly chosen and not inputed by the user. The author tested this algorithm on four unconstrained problems.

We now give an overview on methods that have shown a significant improvement from the filled function methods (see 3.2.1) . These methods solve more problems and are more efficient than filled function methods. These methods are called *auxiliary* function methods.

3.2.2 Auxiliary function methods

We now outline two developed auxiliary function methods. These methods minimize an auxiliary function $T(x, k)$ instead of $f(x)$. We state some definitions that are useful to refer to in this section.

Definition 4. *For any $x \in Z^n$, a set of integer points $N(x) \subset Z^n$ is called a neighbourhood of the integer point x, if $\{x, x + e_i, x - e_i, i = 1, \cdots, n\} \subset N(x)$, where e_i is an n-dimensional Standard Basis Vector.*

Definition 5. *An integer point $x_0 \in X \cap Z^n$ is called a discrete local minimizer of $f(x)$ over $X \cap Z^n$, if $f(x) \geq f(x_0)$, for all $x \in N(x_0) \cap X$.*

Definition 6. *An integer point $x_0 \in X \cap Z^n$ is called a discrete global minimizer of $f(x)$ over $X \cap Z^n$, if $f(x) \geq f(x_0)$, for all $x \in X \cap Z^n$.*

Definition 7. *An integer point $x_0 \in S$ is called a constrained discrete local minimizer of problem (P), if $f(x) \geq f(x_0)$ for all $N(x_0) \cap S$.*

Definition 8. *An integer point $x_0 \in S$ is called a constrained discrete global minimizer of problem (P), if $f(x) \geq f(x_0)$ for all $x \in S$.*

The first auxiliary function based stochastic approximate method for finding a global minimum value of (P) is presented by Zhu and Fan [2]. This method presents an auxiliary function $T(x,k)$ with a parameter $k > 0$, which has the same discrete global minimizers as the original problem (P). Assuming that x_1^* is the current best optimal solution and $G(t)$ is any monotonically increasing function, the auxiliary function can be defined as:

$$T(x,k) = \begin{cases} f(x) + kG(\|x - x_0\|) & \text{if } f(x) \geq f(x_1^*), \\ f(x) & \text{if } f(x) \leq f(x_1^*). \end{cases} \tag{3.2}$$

where $\|\cdot\|$ designates the p-norm, $p = 1, 2$ or ∞, $G(0) = 0$ and $x_0 \in X \cap Z^n$ is a discrete local minimizer of the problem (P). The following nonlinear integer programming problem

$$(AP) \quad \begin{cases} \min & T(x,k) \\ \text{s.t.} & x \in X \cap Z^n \end{cases}$$

is constructed. The main step of this method is solving problem (AP) to find a discrete local minimizer of problem (P) lower than its current best one x_1^*. Some of the properties of $T(x,k)$ are stated below without proof:

Theorem 4. *If x_0 is a discrete local minimizer of the problem (P) with $f(x_0) \geq f(x_1^*)$, then x_0 is a discrete local minimizer of the problem (AP).*

Lemma 5. *For all $x \in S = \{x \in X \cap Z^n : f(x) < f(x_1^*)\}$, and for all $y \in (X - S) \cap Z^n = \{x \in X \cap Z^n : f(x) \geq f(x_1^*)\}$, it holds that $T(x,k) < T(y,k)$.*

Corollary 6. *If x_1^* is not a discrete global minimizer of problem (P), then $\{x \in X \cap Z^n : f(x) < f(x_1^*)\} \neq \phi$, and all discrete global minimizers of problem (AP) are in the set $\{x \in X \cap Z^n : f(x) < f(x_1^*)\}$.*

Theorem 7. *Suppose that x_1^* is not a discrete global minimizer of the problem (P). For $y \in \{x \in X \cap Z^n : f(x) < f(x_1^*)\}$, if y is a discrete local minimizer of problem (AP), then y is a discrete local minimizer of problem (P), and vice versa.*

So by **Corollary 6** and **Theorem 7**, if x_1^* is not a discrete global minimizer of problem (P), then problems (P) and (AP) have the same global minimizers and global minimal values.

It can be clearly seen that the landscape of $T(x,k)$ on $\{x \in X \cap Z^n : f(x) < f(x_1^*)\}$ is not dependent on parameter k, since by equation (3.2), for all $\{x \in X \cap Z^n : f(x) < f(x_1^*)\}$, $T(x,k) = f(x)$. However, the landscape of $T(x,k)$ on $\{x \in X \cap Z^n : f(x) \geq f(x_1^*)\}$ is dependent on k and $G(\|x - x_0\|)$.

Some of the properties of $T(x,k)$ dependent on k are :

Lemma 8. *For any $x = (x_1, \cdots, x_n)^T \in X \cap Z^n$, if $x \neq (x_{01}, \cdots, x_{0n})^T \in X \cap Z^n$, then there exists $y = (y_1, \cdots, y_n) \in N(x) \cap X$ such that $G(\|y - x_0\|) < G(\|x - x_0\|)$.*

Theorem 9. *For the function $T(x,k)$, we have the following results.*

1. *For any $x \in S_1 = \{x \in X \cap Z^n : f(x) \geq f(x_1^*)\}$, if there exists $y \in N(x) \cap X$ such that $f(y) < f(x_1^*)$, then x is not a discrete local minimizer of problem (AP).*

2. *For any $x_1 \in S_1 = \{x \in X \cap Z^n : f(x) \geq f(x_1^*)\}$, $x \neq x_0$, let*
   $$A(x) = \max\left\{0, \min_{\substack{z \in N(x) \cap X \\ \|z - x_0\| < \|x - x_0\|}} \frac{f(z) - f(x)}{G(\|x - x_0\|) - G(\|z - x_0\|)}\right\}.$$
   If $k > A(x)$, then x is not a discrete local minimizer of the problem (AP).

3. *Especially, if*
   $$k > \max_{x \in X \cap Z^n} A(x), \tag{3.3}$$
   then for all $x \in S_1 = \{x \in X \cap Z^n : f(x) \geq f(x_1^*)\}$, $x \neq x_0$, x is not a discrete local minimizer of problem (AP).

Assertions 2 and 3 of **Theorem 9** suggests that if minimization of $T(x,k)$ gets stuck at a discrete local minimizer in the set $\{x \in X \cap Z^n : f(x) \geq f(x_1^*)\}$, then by increasing the value of k sufficiently, minimization of $T(x,k)$ can escape from the discrete local minimizer.

Moreover, by the proof of **Theorem 9** (see [2]), it is concluded if k satisfies the third assertion of Theorem 9, then $T(y,k) < T(x,k)$ for all $y \in N(x) \cap X$ such that $\|y - x_0\| < \|x - x_0\|$. This implies that if k satisfies (3.3), then while minimizing $T(x,k)$ from any initial point in $X \cap Z^n$, the minimization sequence will converge to the prefixed discrete local minimizer x_0, or converge to a discrete local minimizer in the set $\{x \in X \cap Z^n : f(x) \leq f(x_1^*)\}$.

Definition 9. *Suppose that x is a discrete local minimizer of $f(x)$ over $X \cap Z^n$. $R(x) \subset X \cap Z^n$ is called a discrete attraction region of x, if starting from any initial point in $R(x)$ to minimize $f(x)$ on $X \cap Z^n$ using the local search method, i.e., DLS, will converge to x.*

If any point y is a discrete local minimizer of problem (P), then, by **Theorem 9**, to escape from the discrete attraction region of y by minimizing $T(x,k)$, k should be large enough.

Suppose there exists an arbitrary case such that the following conditions are satisfied: $x \in X \cap Z^n$, $f(x) > f(x_1^*)$ and $f(y) \geq f(x_1^*)$ for all $y \in N(x) \cap X$. Also, suppose there exists $z \in N(x) \cap X$ such that $\|z - x_0\| > \|x - x_0\|$, $f(z) < f(x)$, and z is in the discrete attraction region of a point lower than x_1^*. In such a case, if the value of k is too big such that equation (3.3) holds, then **Theorem 9** implies that $T(z,k) > T(x,k)$. Using DLS to solve problem (AP) starting from x will leave z to an integer point nearer to x_0, but not into the discrete attraction region.

So there is one question : whether or not the minimization of $T(x,k)$ on $X \cap Z^n$ starting from x could move to z. That is, how do we choose the value of k such that $T(z,k) < T(x,k)$ if $f(z) < f(x)$?. In fact, the following result was presented.

Theorem 10. *Suppose that $z \in N(x) \cap X$, and $f(x) > f(z) \geq f(x_1^*)$. Then $T(z,k) < T(x,k)$ if and only if one of the following conditions holds:*

1. $k = 0$.
2. $k > 0$ *and* $\|z - x_0\| \leq \|x - x_0\|$.
3. $k > 0$, $\|z - x_0\| > \|x - x_0\|$, *and* $k < \frac{f(x)-f(z)}{G(\|z-x_0\|)-G(\|x-x_0\|)}$.

Theorem 10 implies that in some cases $T(x,k)$ could not keep the descent points of $f(x)$ in the region $\{x \in X \cap Z^n : f(x) \geq f(x_1^*)\}$, if k is too large. This implies that while minimizing

the auxiliary function $T(x,k)$ from an initial point in the discrete attraction region of a discrete local minimizer of $f(x)$ lower than x_1^* (the current best local minimizer), if one wants to find a discrete local minimizer lower than x_1^*, k must not be too large.

There now seems to be a contradiction since, by **Theorem 9**, in order to bypass a previously converged discrete local minimizer, the value of k should be large enough. So in the algorithm that is presented next, while minimizing $T(x,k)$ on $X \cap Z^n$, we take $k = 0$ initially, and increased the value of k sequentially. The formal algorithm is called *Dynamic convexized method* and is given as follows:

Algorithm 3: Dynamic convexized method (DC) [2]

Step 1. Select randomly a point $x \in X \cap I^n$, and start from which to minimize $f(x)$ on $X \cap Z^n$ using Algorithm 1 to get a discrete local minimizer x_1^* of problem (P). Let N_L be a sufficiently large integer, and let δ_k be a positive number. Set $N = 0$.

Step 2. Select a point $x_0 \in X \cap I^n$, such that x_0 is a discrete local minimizer of problem (P) and $f(x_0) \geq f(x_1^*)$. Construct a function $T(x,k)$ with k, x_1^* and x_0.

Step 3. Set $k = 0$, and $N = N + 1$. If $N \geq N_L$, then go to Step 6; otherwise draw randomly an initial point $y \in X \cap I^n$ and go to Step 4.

Step 4. Minimize $T(x,k)$ on $X \cap I^n$ from y using DLS. Suppose that x' is an obtained discrete local minimizer.
If $x' \neq x_0$ and $f(x') \geq f(x_1^*)$, then set $k = k + \delta_k$, $y = x'$, and repeat Step 4.
If $x' = x_0$, then go to Step 3.
If $f(x') < f(x_1^*)$, then go to Step 5.

Step 5. Let $x_1^* = x'$, and go to Step 2.

Step 6. Stop the algorithm, output x_1^* and $f(x_1^*)$ as an approximate global minimal solution and global minimal value of problem (P) respectively.

The explanation of the above mentioned algorithm is as follows: Take $k = 0$ initially, and a starting point in $X \cap Z^n$ is taken randomly to minimize $T(x,k)$ using DLS. If the minimization

sequence converges to a point $x' \neq x_0$ and $f(x') \geq f(x_1^*)$, then the value of k is increased, and $T(x,k)$ is minimized on $X \cap Z^n$ from x'. If at this time the minimization sequence converges to a point $x'' \neq x_0$ and $f(x'') \geq f(x_1^*)$, then by **Theorem 9**, the value of k is too small, hence the value of k is increased and $T(x,k)$ is minimized on $X \cap Z^n$ from x'' again, till the minimization sequence converges to x_0 or a point in $\{x \in X \cap Z^n : f(x) < f(x_1^*)\}$.

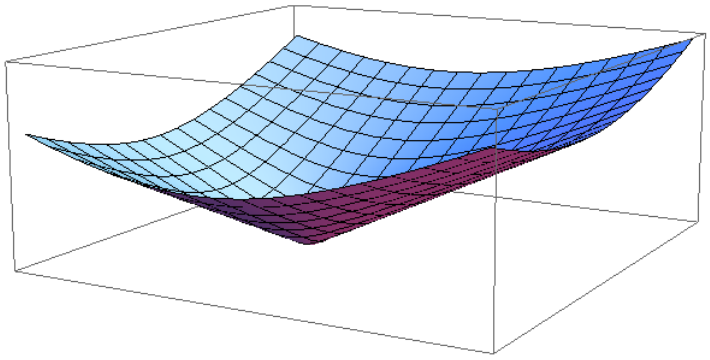


Figure 3.1: An example of the function $G(t)$ used in equation (3.2)

If the minimization sequence converges to x_0, then the process above is repeated. If the minimization sequence converges to a point in $\{x \in X \cap Z^n : f(x) < f(x_1^*)\}$, then by **Theorem 7**, a discrete local minimizer of (P) lower than x_1^* has been found. The point x_1^* is reset, and the above process is repeated. This may be clarified by the following graphical explanation. Suppose that $G(x) = \|x - x_0\|^2$ represents G in equation (3.2), where $x = [x_1, x_2]$, $x_1, x_2 \in [0, 10]$ and $x_0 = [4, 4]$. The function G is the cone given in Figure 3.1. Suppose that the objective function $f(x)$ is the top-left diagram in Figure 3.2 marked $k = 0$. By iteratively increasing k in steps of δ_k, we can see how the function $f(x)$ changes its landscape. When $k = 20$ we still see that there exists many local minima so this means that the value of k is too small and hence it must be increased. When the value of k is 40, we see that the landscape of the function T has a definite bias towards the point $[4, 4]$. And finally, when k has reached the value of 80, we see that doing a local search routine would lead to the point of interest $x_0 = [4, 4]$.

If problem (P) is constrained, it is converted into an equivalent unconstrained NIP problem by using a penalty function approach presented by Sinclair [9], thereafter the auxiliary function method is applied to the unconstrained problem. The method minimizes $T(x,k)$ using a discrete local search from randomly generated starting points. The minimization of $T(x,k)$ using a

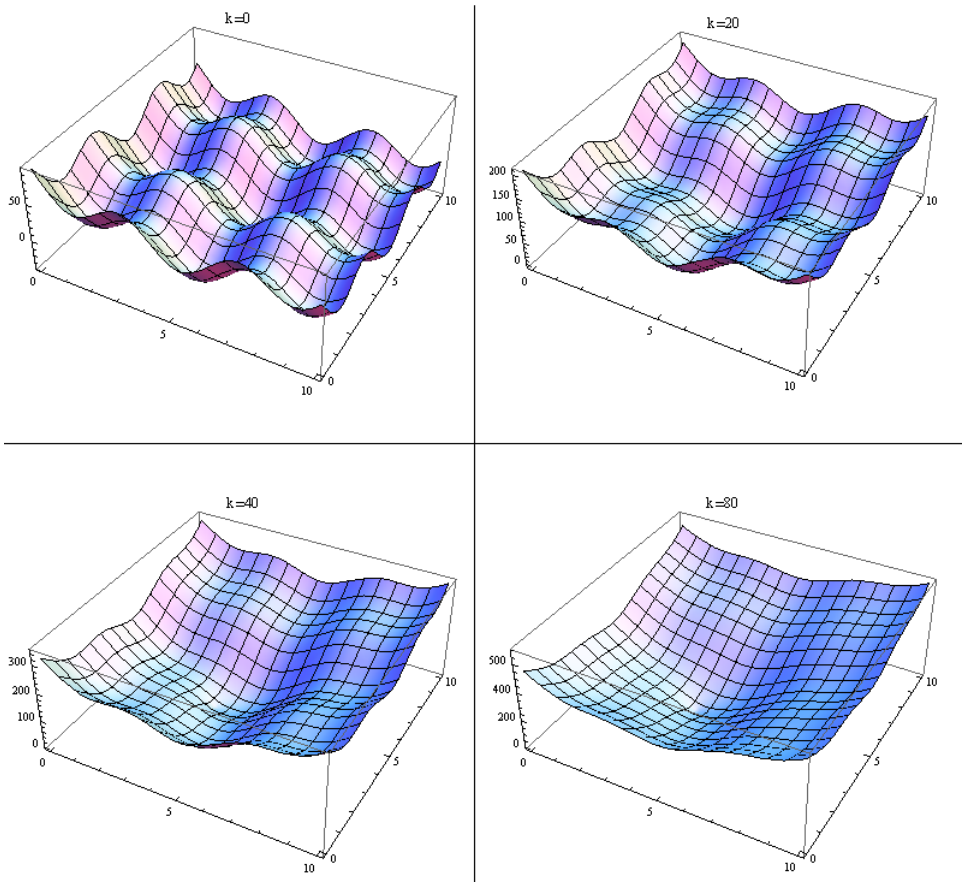


Figure 3.2: Effects of increasing the parameter k in $T(x, k)$

discrete local search method can successfully escape from previously converged discrete local minimizers. Zhu and Fan [2] have proved that this method converges asymptotically with probability one and they have tested it on a wide range of test problems. Next, we discuss an extension of this approach which was proposed by Zhu and Ali [6].

The extension suggested by Zhu and Ali [6] (for constrained problems) is a change in the auxiliary function as follows:

Let x_1^* be the current best minimal solution of problem (P), and let f_1^* be a finite number such that if x_1^* is a feasible integer point of (P) then $f_1^* = f(x_1^*)$; otherwise f_1^* is an upper bound on the global minimal value of (P). Take

$$p(x) = \alpha[\max\{0, g_i(x), i \in K\} + \sum_{j \in J} |h_j(x)|] \tag{3.4}$$

where α is any positive number. Obviously, $x \in X \cap Z^n$ is a feasible solution of the problem (P) if and only if $p(x) = 0$. The auxiliary function is given by

$$T(x,k) = \begin{cases} \max\{0, f(x) - f_1^*\} + p(x) + k\|x - x_1^*\| & \text{if } f(x) \geq f_1^* \text{ or } p(x) > 0, \\ f(x) - f_1^* & \text{otherwise,} \end{cases} \tag{3.5}$$

where k is a nonnegative parameter, $\|.\|$ designates the Euclidean norm, e.g. the 2-norm.

Let S be the set of feasible integer points of the problem (P), i.e.
$S = \{x \in X \cap Z^n : g_i(x) \leq 0, i \in K, h_j(x) = 0, j \in J\}$. This leads to the following definitions:

Definition 10. *An integer point $x_0 \in S$ is called a constrained discrete local minimizer of problem (P), if $f(x) \geq f(x_0)$, for all $x \in N(x_0) \cap S$.*

Definition 11. *An integer point $x_0 \in S$ is called a constrained discrete global minimizer of problem (P), if $f(x) \geq f(x_0)$, for all $x \in S$.*

It is easy to conclude that a constrained discrete global minimizer of problem (P) is a constrained discrete local minimizer of problem (P). The following constrained local search method for problem (P) was presented.

Constrained local search (CLS) [6]

Step 1. Given an initial feasible integer point x_0 of problem (P).

Step 2. If x_0 is a constrained discrete local minimizer of problem (P), then stop; else take an integer point $x \in N(x_0) \cap S$ such that $f(x) < f(x_0)$.

Step 3. Let $x_0 := x$, and go to step 2.

It is generally known that finding a constrained discrete local minimizer of problem (P) is difficult. A constrained discrete local minimizer of problem (P) is a feasible solution, which satisfies that $g_i(x) \leq 0, i \in K, h_j(x) = 0, j \in J, x \in X \cap Z^n$. It has been found to be NP-hard to find feasible solution of this inequality system. The formal algorithm for this method is called

Dynamic Convexized Method [6] and is stated as follows:

Algorithm 4: Discrete dynamic convexized method for constrained problems [6]

Step 1. Let f_1^* be a large number such that it is an upper bound on the global minimal value of problem (P). Select randomly a point $x \in X \in Z^n$, and start minimization of $\max\{0, f(x) - f_1^*\} + p(x)$ from x over $X \cap Z^n$ using DLS to get a discrete local minimizer x'. If $p(x') > 0$, then let $x_1^* = x'$; otherwise use CLS to minimize $f(x)$ over S from x' to get a constrained discrete local minimizer of problem (P), denote it as x_1^*, and let $f_1^* = f(x_1^*)$. Let N_L be a sufficiently large integer, and let δ_k be a positive number. Set $N = 0$.

Step 2. Set $k = 0$, and $N = N + 1$. If $N \geq N_L$, then go to Step 5; otherwise draw uniformly at random an initial point $y \in X \cap Z^n$ and go to Step 3.

Step 3. Minimize $T(x, k)$ over $X \cap Z^n$ from y using DLS. Suppose that x' is an obtained discrete local minimizer.
If $x' \neq x_1^*$ and $f(x') \geq f_1^*$ or $p(x') > 0$, then set $k = k + \delta_k$, $y = x'$, and repeat Step 3.
If $x' = x_1^*$, then go to Step 2.
If $f(x') < f_1^*$ and $p(x') = 0$, then go to Step 4.

Step 4. Let $x_1^* = x'$, and go to Step 2.

Step 5. Stop the algorithm, if $p(x_1^*) = 0$, then output x_1^* and $f(x_1^*)$ as an approximate global minimal solution and global minimal value of problem (P), respectively.

The basic idea of this algorithm is as follows: Initially, k is taken to be zero and a starting point is randomly selected in $X \cap Z^n$ to minimize $T(x, k)$ on $X \cap Z^n$ using DLS. If the minimization sequence converges to a point $x' \neq x_1^*$, and $f(x') \geq f_1^*$ or $p(x') > 0$, then the value of k is increased, and $T(x, k)$ is minimized on $X \cap Z^n$ from x'. If at this time the minimization sequence to a point $x'' \neq x_1^*$, and $f(x'') \geq f_1^*$ or $p(x'') > 0$, then the value of k is deemed too small, then the value of k is increased and $T(x, k)$ on $X \cap Z^n$ is minimized from x'' again, till the minimization sequence converges either to x_1^* or to a point in $\{x \in X \cap Z^n : f(x) < f_1^*, \text{and} \quad p(x) = 0\}$.

If the minimization sequence converges to x_1^*, then the above process is repeated. If the minimization sequence converges in $\{x \in X \cap Z^n : f(x) < f_1^*, \text{and} \quad p(x) = 0\}$, then a constrained discrete local minimizer of (P) better than x_1^* exists. The authors let x_1^* be the better constrained discrete local minimizer found, and then the above process is again repeated.

Two reformulation-based methods, presented in [2] and [6], seem to be able to solve (P) for a variety of problems. One of the strengths of the reformulation-based approach presented in [2, 6] is that they solve a wide range of bound constrained and nonlinearly constrained nonlinear problems. However, these methods are inefficient due to the local search procedure and multi-start method mentioned in an earlier chapter. Furthermore, we think that Step 4 of the 'Discrete dynamic method' [2] may seem to waste its effort by trying to escape a local minimum by biasing the search to a previously converged local minimum x_0. In this research, we aim to improve the local search as well as the way an algorithm navigates the search space X instead of using the multi-start procedure.

3.3 Darwin and Boltzmann mixed strategy

Tian et.al [29] introduced an interesting method based on the *Darwin* and *Boltzmann* strategies for solving unconstrained NIP problems which we present next.

3.3.1 Nonlinear integer programming by Darwin and Boltzmann mixed strategy

This section presents a general stochastic iterative algorithm for NIP problems. The algorithm synthesizes the advantages of the *Darwin strategy* and the *Boltzmann annealing strategy* [29].

Most complex systems observed in nature and society are the results of an evolutionary process. The two well-known evolutionary processes are Darwinian processes and irreversible thermodynamic processes. It is believed that the strategies developed in these evolutionary processes have important values for the theory and practice of modem science and technology.

The first strategy used in this method is the Darwin strategy which contains the following elements:

1. self-reproduction of good species that shows maximal fitness;

2. mutation processes that change the phenotypic properties of the species;

3. increase of the precision of self-reproduction and mutation in time.

The second strategy used was the Boltzmann annealing strategy, which includes the following elements:

1. motion along gradients to reach steepest descent;

2. stochastic thermal motion to avoid locking in local minima;

3. decrease of the temperature to increase the precision of the search.

The method presented by Tian et.al [29] is based on the simulated annealing method [8]. The main reason for this is because simulated annealing escapes from local minima while it still maintains the favorable features of simplicity and general applicability for the local search algorithms. Given the problem (P) without any constraints, we next present the DBMS (Darwin and Boltzmann Strategy) algorithm [29]:

Algorithm 5: Darwin and Boltzmann mixed strategy for NIP problem (DBMS)

Step 1. Given an initial solution $x \in S$;
Setting an iterative counter $k = 0$ for M_k.
Given simulation mutation parameters $M_0 > 0$;

Step 2. Mutation process: performing the following Boltzmann strategy.

Step 2.1. Generating a random solution $x' \in N(x)$; Evaluating $\Delta f = f(x') - f(x)$.

Step 2.2. If 'Metropolis criterion' is satisfied, i.e.,$\min\{1, \exp(-\Delta f/M_k)\} > \eta \in [0, 1)$, then $x = x'$.

Step 2.3. If 'Metropolis equilibrium' under M_k is realized, then go to Step 3; else go to Step 2.1.

Step 3. Self-reproduction process: doing the following self-reproduction evolution steps.

Step 3.1. Evaluating $f(x)$.

Step 3.2. Giving a temporary set $V = \{x\}$, and setting $f_{pre} = f(x)$.

Step 3.3. Picking a solution $x' \in (N(x) \setminus V)$, and $V = V \bigcup \{x'\}$; Evaluating $\Delta f = f(x') - f(x)$.

Step 3.4. If $\Delta f < 0$, then $x = x'$.

Step 3.5. If $(N(x) \setminus V) = \Phi$ (null set), then go to Step 3.6; else go to Step 3.3.

Step 3.6. If $f(x) < f_{pre}$, then go to Step 3.2; else go to Step 4.

Step 4. Annealing process: reducing simulation mutation parameter $M_{k+1} = M_k - \Delta M_k, \Delta M_k > 0$.

Step 5. If 'stop criterion' is not satisfied, i.e. $M_k > M$, then setting $k = k + 1$, go to Step 2; else outputting $x_{opt} = x$.

The above algorithm combines the Boltzmann annealing process with the mutation process and mutation parameter adjustment of the Darwin strategy. An optimal search process is then performed repeatedly by the self-reproduction processes of the Darwin strategy and the Boltzmann annealing strategy. The self-reproduction processes use steepest descent searching for local minima. The mutation process performs the 'climbing hill' courses of the Boltzmann strategy. It always starts with some local minimums, and then escape from local optimal 'traps'. The annealing process constantly decreases the mutation parameters until it reaches the global optimum.

The authors proved that the DBMS algorithm converges and runs in a polynomial time. This analysis was achieved using the theory of finite Markov chains since DBMS is a stochastic iterative algorithm. The algorithm was tested on three test problems of a dimension 5, 6, 7 respectively.

We conclude by stating that this method still needs rigorous testing on more test problems before commenting on its robustness and efficiency. Now that we have given an extensive review on algorithms that solve problem (P) and highlighted their disadvantages, we now move on to the following chapter in which we will present a new algorithm that will counter these disadvantages.

Chapter 4

Voronoi diagrams and the RRT algorithm

In this chapter, we present a rapidly-exploring random tree (RRT) based algorithm. Central to the RRT algorithm is the voronoi diagram. We begin with a discussion of Voronoi diagrams before presenting the detailed RRT algorithm.

4.1 Voronoi diagrams

A *Voronoi diagram* is a decomposition of a metric space determined by distances to a specified discrete set of points in the space. A Voronoi diagram of a set of points partition the space into a set of convex polygons so that each polygon contains exactly one point of the set [14, 19]. The Russian mathematician Georgy Fedoseevich was the first to define and study the general n-dimensional case of Voronoi diagrams in 1908. However, the informal use of the diagram can be traced as far back as 1664 where they were utilized by Descartes [19]. Thereafter, Dirichlet used 2-D and 3-D Voronoi diagrams in his study of quadratic forms. An illustration of how the majority of people who died in the Soho cholera epidemic was showcased by the use of a Voronoi diagram in 1854 by a British physician called John Snow.

Voronoi diagrams have been used across many scientific fields. Applications go as far wide as town planning, climatology, computer graphics, autonomous robot navigation and computational chemistry to name a few. We will give an in-depth analysis of Voronoi diagrams in this

section and attempt to find a use for them in our quest to solve global optimization problems.

4.1.1 Computing Voronoi diagrams

Without loss of generality, given a set S of n points in R^d (called sites), the Voronoi diagram of S is a partition of space into cells, such that each cell is in the region of space consisting of all points that are closer to a particular site than any other [17]. To clarify this concept, we use a diagram to aid the explanation. Each site $s \in S$ in Figure 4.1 has a *Voronoi cell*, also called

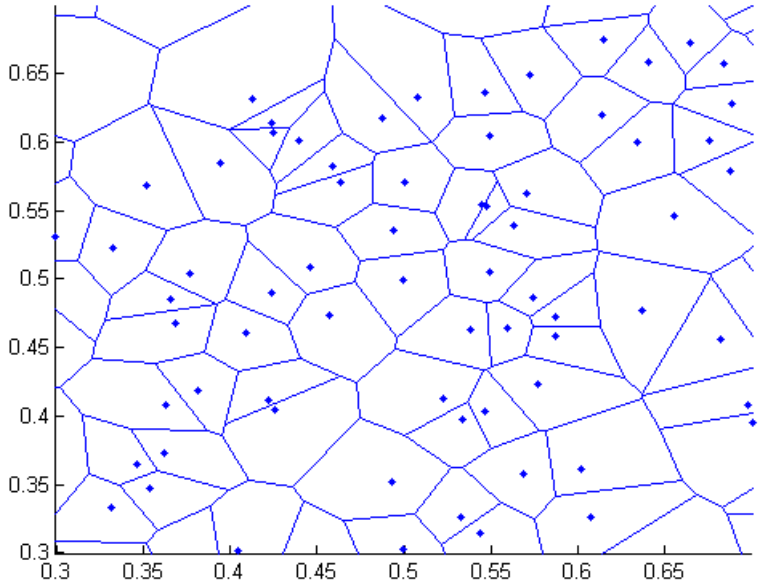


Figure 4.1: The Voronoi diagram of a random set of points S in the plane.

a Dirichlet cell, consisting of all points closer to s than to any other site. The segments of the Voronoi diagram in Figure 4.1 are all the points in the plane (line in two dimensions) that are equidistant to the two nearest sites. If the set S only contains two points (sites), say x and y, then the set of all points equidistant from x and y is a hyperplane. The hyperplane is the boundary between the set of all points closer to x than to y, and the set of all points closer to y than to x. Some basic **properties** of Voronoi diagrams are listed below:

- The closest pair of points corresponds to two adjacent cells in the Voronoi diagram (see Figure 4.1).

- The dual graph for a Voronoi diagram corresponds to the *Delaunay triangulation*[1] for the same set of points in S.
- Two points are adjacent on the convex hull if and only if their Voronoi cells share an infinitely long side.

Using the concpet of Voronoi diagrams to solve problem (P) does not seem far-fetched in our initial observation. Using the multi-start technique (see Chapter 2), one can generate Voronoi diagrams in every iteration and set a stopping criterion based on the largest Voronoi cell. Points that were 'promising' would be minimized by applying a local search technique then an approximate global minimizer could be found. But some crucial information was unearthed which led to this idea's downfall. First, Voronoi diagrams work considerably well for two and three dimensional data sets only but unfortunately many applications are of higher dimensions. The complexity of Voronoi diagrams can be as high as $N^{\frac{n}{2}}$ (n being the dimension of the points in the set) [17]. Also, constructing Voronoi diagrams can be complicated due to either numerical inaccuracies or degeneracies as a result of cocircular points. And finally, finding the size of a Voronoi cell proved to be challenging and tedious (especially in large dimensions). Intuitively, Voronoi diagrams implicitly encode information of what site (point) is closest to a given point. This implies that the speed of computing Voronoi diagrams depends on the nearest neighbour algorithm being used.

The shortcomings of the previous paragraph have led to a great deal of research in order to implement simpler structures that can be used in place of Voronoi diagrams [14, 17]. One such structure is named *approximate voronoi* [11]. The readers may read further about approximate voronoi if they wish. Although approximate voronoi improves the original idea substantially, we also encounter the problem of computing the size of each Voronoi cell. Hence this leads us away from the idea of using the Voronoi diagrams to solve problem (P). This conclusion leads us to *rapidly-exploring random trees* commonly know as *RRTs*.

[1] A Delaunay triangulation for a set S of points in the plane is a triangulation such that no point in S is inside the circumcircle of any triangle in the Delaunay triangulation. Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation, they tend to avoid skinny triangles. This concept is widely applied in computational geometry.

4.2 Rapidly-exploring random trees (RRT)

4.2.1 Introduction

Rapidly-exploring random tree is a method that incrementally constructs a search tree that attempts to *rapidly* and *uniformly* explore any search space. The method was developed by LaValle and Kuffner [13]. The tree is constructed in such a way that any sample in the space is added by connecting it to the closest sample already in the tree. RRT's are widely used in the field of robotics for motion planning [13, 16, 21]. We now discuss the problem formulation for general RRTs for motion planning.

4.2.2 Problem formulation

The RRT algorithm is a randomized algorithm useful for exploring large state spaces that cannot be searched exhaustively.

We now explain how the general RRT algorithm works with the aid of Figure 4.2. The RRT

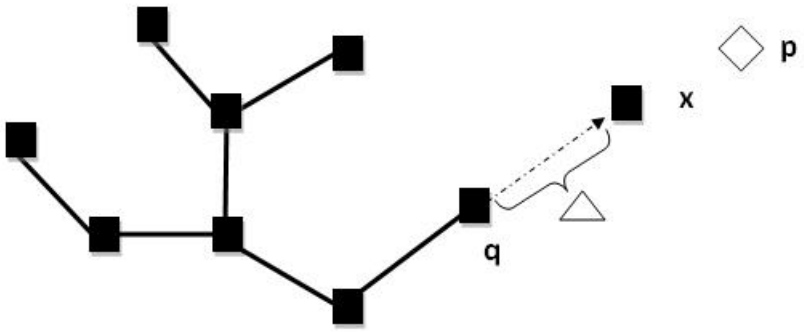


Figure 4.2: An example of an RRT extension.

algorithm iteratively chooses a random point p in the state space and attempts to extend, by some distance Δ, the current search toward that point. The iterative random points can be

generated using a distribution[2] over the state space, but would include some bias toward a goal [3]. Considering the random point p, the extension of the RRT algorithm is performed from the node q to x only if the q is the nearest neighbour of p. So this intuitively means that p must lie in the Voronoi region of q and the nodes on the frontier of the search tree generally have the largest Voronoi regions. Now, in the context of motion planning, an action u must observe any dynamic constraints on the robot. So when the action u is applied to the robot at state q, if all the dynamic constraints are satisfied, then this results in some new state x [4]. Then once extension has been made to point x, without violating any of the constraints, the new state x and the action u are added to the tree. The pseudo codes of the supporting functions are presented below.

Algorithm 6: The RRT algorithm

Procedure 1: to generate a random point p and add it to the tree.

BUILD_RRT(x_{init})

1. T.init(x_{init})
2. **while** ($x_{goal} \notin T$) **do**
3. $\{p, q\} \leftarrow$ SELECT_EXTENSION(T);
4. EXTEND(T, p, q)
5. Return T

---

[2]The distribution of the random function may not necessarily be normal.

[3]In motion planning, a robot navigates through a space towards a pre-defined destination (goal)

[4]It is important to note that x is in the direction of p from the state q, but x and p are not expected to coincide.

Procedure 2: to generate a random point p and calculate the nearest neighbour (q) to p on the tree T.

SELECT_EXTENSION(T)

1. $p \leftarrow$ RANDOM_STATE();
2. $q \leftarrow$ NEAREST_NEIGHBOUR(p, T);
3. Return T

Procedure 3: to add the vertex x to the tree T.

EXTEND(T, p, q)

1. **if** (NEW_STATE(p, q, x, u)) then
2. T.add_vertex(x)
3. T.add_edge(q, x, u);

Remark 3. *Procedure 1 is the main function of the RRT algorithm. It uses both procedure 2 and 3 to create and extend the RRT tree T where an initial random point x_{init} is used as the input. Procedure 2 selects a random point p and finds the nearest neighbour (q) of point p in the tree T. Procedure 3 extends the tree T from node q towards p by Δ to find a new node x.*

4.2.3 Properties of the RRT

This section highlights some key properties presented by LaValle and Kuffner [13] and LaValle [18] that suggest why the RRT algorithm is an effective tool in search space exploration. The key advantages of RRT's are:

1. the expansion of an RRT is heavily biased toward unexplored portions of the state space;

2. the distribution of the vertices in an RRT approaches the sampling distribution, leading to consistent behavior;

3. an RRT is probabilistically complete under very general conditions;

4. the RRT algorithm is relatively simple, which facilitates performance analysis;

5. an RRT always remains connected, even though the number of edges are minimal;

6. an RRT can be considered as a path planning module, which can be adapted and incorporated into a wide variety of planning systems;

7. entire path-planning algorithms can be constructed without requiring the ability to steer the system between two prescribed states (e.g. the initial state and final state), which greatly broadens the applicability of RRT's.

To visualize the RRT algorithm in work, we consider the case where $X = [0, 10^5] \times [0, 10^5]$ is the bounded search space in the plane. Let ρ represent the Euclidean metric. Figure 4.3 below illustrates an RRT at different stages in its implementation when the RRT starts at an initial point $x_{init} = [50000, 50000]$.

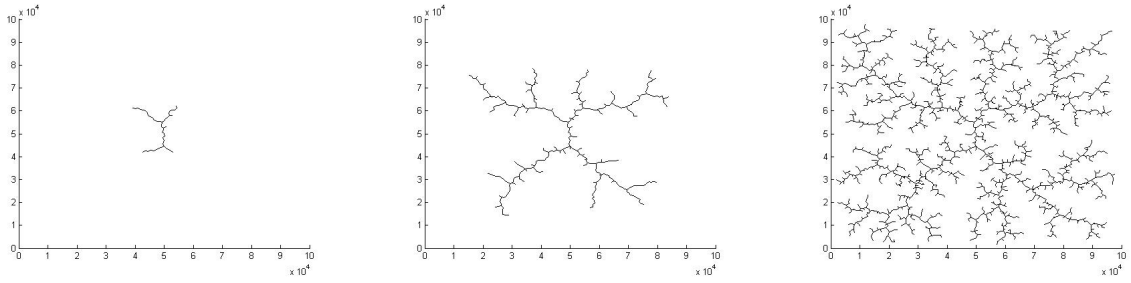


Figure 4.3: RRT algorithm at 3 different stages

The first diagram in Figure 4.3 shows how the RRT quickly expands in a few directions after 50 iterations of the algorithm, to explore the four corners of the search space. The second diagram in Figure 4.3 shows how the tree has grown by searching the state space uniformly. Unlike a random walk, the RRT works by being biased towards places that have not yet been visited. This may be observed when the Voronoi diagram of the RRT vertices are computed.

Suppose the Voronoi diagram of the RRT tree is the one depicted in Figure 4.2 (see section 4.1). Some Voronoi cells are larger than others. If a point is chosen randomly in the search space, it is more likely to fall inside the larger Voronoi cells than in the smaller ones. Hence, the larger cells will have a greater probability of being subdivided. This case is shown by the third diagram of Figure 4.3 after 2000 iterations. Usually, an RRT is constructed by iteratively breaking up larger Voronoi regions into smaller ones. Note that the probability that a vertex is selected for extension is proportional to the area of its Voronoi region. This biases the RRT to rapidly explore the search space.

4.2.4 Analysis of RRT's

Some analysis on RRTs have been shown by authors LaValle and Kuffner [13] and LaValle [18]. Suppose that X_{free} is the space that a robot can navigate the search space X without colliding into any objects, and x_{goal} is a target (pre-defined) for which the robot is headed towards. Let $D_k(x)$ denote a random variable whose value is the distance of x to the closest vertex G, in which k is the number of vertices in an RRT. Futhermore, let d_k denote the value of D_k and ϵ denote the incremental distance travelled in the 'EXTEND' procedure (in other words the step size Δ of the RRT). The following lemmas and the theorem below are presented without proofs. The proofs of these lemmas and theorem can be found in [18].

Lemma 11. *Suppose that X_{free} is a convex, bounded, open, n-dimensional subset of an n-dimensional state space. For any $x \in X_{free}$ and positive constant $\epsilon > 0$, $\lim_{k\to\infty} P[d_k(x) < \epsilon] = 1$.*

Lemma 12. *Suppose that X_{free} is a nonconvex, bounded, open, n-dimensional connected component of an n-dimensional state space. For any $x \in X_{free}$ and positive real number $\epsilon > 0$, then $\lim_{n\to\infty} P[d_n(x) < \epsilon] = 1$.*

Theorem 13. *Suppose that x_{init} and x_{goal} lie in the same connected component of a nonconvex, bounded, open, n-dimensional connected component of an n-dimensional state space. The probability that an RRT constructed from x_{init} will find a path to x_{goal} approaches one as the number of RRT vertices approaches infinity.*

Lemma 11 establishes that the RRT will (converge in probability) come arbitrarily close to any point in a convex space. The above lemmas and the theorem establish the probabilistic completeness of the RRT algoithm [13].

Many journal articles have been written on improving the basic RRT algorithm by the way of *biasing*. Urmson and Simmons [15] provided a method for biasing the growth of an RRT based on cost discovered through exploration of the search space. They claim that doing so provides information to the algorithm that allows it to operate in more than just an exploratory manner. LaValle and Kuffner [13] developed an improved version of RRT called *RRT-GoalBias*. The authors pointed out that without any bias toward the goal, convergence is often slow. The RANDOM_STATE function in the basic RRT algorithm (see Algorithm 6 on page 44) is replaced with a function that tosses a biased coin to determine what should be returned. This function biases the RRT as follows:

- If the coin toss results in a 'heads', the point p (in Procedure 2 of the RRT algorithm) is assigned the value x_{goal}.
- Otherwise, if the coin toss results in a 'tails', the point p (in Procedure 2 of the RRT algorithm) is assigned a random value.

It was concluded that in general, it is better to replace the RANDOM_STATE function with a sampling scheme that draws states from a nonuniform probability density function that has a 'gradual' bias toward the goal.

LaValle and Kuffner [13] also suggested the approach of having *bidirectional planners* which was inspired by some research on bidirectional search techniques. This involves growing two RRTs, one from x_{init} and the other from x_{goal}. In each iteration, one tree is extended, and an attempt is made to connect the nearest vertex of the other tree to the new vertex. Intuitively, a solution is found if the two RRTs meet. Through many experiments and a wide range of examples, it was concluded that, when applicable, the bidirectional approach is much more efficient than a single RRT and hence an improved performance is obtained by growing two trees. Henceforth a natural question arose: if growing two trees is better than one, how about growing three or more RRTs? Having more trees brings about a complicated decision problem. The computational time must be divided between attempting to explore the space and attempting

to connect RRTs to each other. It is also not clear which connections should be attempted. Many research issues remain in the development of this and other RRT-based planners.

Having considered all the information in this section, based on the advantages of the RRT algorithm as a search space exploration tool, we think that it can be used as part of a global optimization algorithm to solve problem (P). Instead of using the multi-start technique as a base to an optimization algorithm (such as in [2, 6]), we see it more appropriate to use an RRT to explore new regions of the space X. This will ensure that the search has a structure which is significantly better that a random walk. The idea of using RRTs as a search space exploration tool has never been introduced in the field of global optimization. We think that this will assist the search to efficiently find the global minimum, x^*, of problem (P). The results of the RRT-based global optimization algorithms are presented in chapter 5.

4.3 RRT-based optimizer

The algorithm we now present implements the concept of RRT together with the *topographical clustering* technique to find the global minimum value of problem (P). We call the algorithm RRT-optimizer and denote it by RRTOpt. An overview of RRTOpt algorithm is as follows:

- Select a point $p \in X \bigcap Z^n$ to find a point x to add the vertex to the RRT tree (as in Figure 4.2).
- If the number of nodes in the RRT tree (T) are greater than a specified value, keep track of the last six Δ values (in Procedure 3 of the RRT algorithm) in an array dt.
- If the average of dt, denoted v, is less than (SC), perform topographical clustering on the best k nodes using the parameters N and g_m[5].
- Perform a local search routine on all the graph minimizers[6].
- Output the best local minimizer, x_L, found.

[5]For the description of N and g_m, see chapter 2.5 page 18.

[6]For each of the k nodes N neighbouring points are formed. Graph minimizers are found using $N+1$ points and local searches are performed from each graph minimizer.

At each iteration the random starting point p implements a nearest neighbour search of all the vertices in the RRT tree. Then the point in the tree which is closest to p is extended towards p by some distance Δ to give x which is then added to the RRT tree (T) (see Figure 4.2). For the purpose of stopping RRTOpt, we keep record of the last six extension distances (Δ's) in array dt. If the average, v, of dt is less that some pre-defined stopping parameter SC, then the algorithm is stopped and the best point or local minimizer found so far, x^*, is given as the approximate global minimizer to problem (P). One must note at this point that at each iteration, the RRT algorithm may produce two points, p and x (see Figure 4.2). If the extension distance Δ is greater than the distance between x and p, then $x = p$ otherwise; x is an extension from the nearest point in the tree. This makes $D_k(x)$ random (see subsection 4.2.4). The point p is saved in the array $xbest$ if it has yielded a lower function value than any of the best k nodes in $xbest$. The RRTOpt algorithm is stopped when v reaches some given value, e.g. $v \leq SC$.

We now present the formal RRTOpt algorithm.

Algorithm 7: The RRTOpt algorithm

Step 1 . Set parameters N,$g_m, k, r, init_points$ and Δ. Initialize counters $i = 1$ and v to zero, and input the value SC.

Step 2. Select randomly a point $p \in X \bigcap Z^n$ and set this as the root of the RRT by calling the function BUILD_RRT(P).

Step 3. Select randomly a point $p \in X \bigcap Z^n$ and extend the RRT tree using the SELECT_EXTENSION(T) function to add the point x to the RRT tree.

Step 3.1 If $i < init_points$, go to repeat Step 3. Otherwise go to Step 4.

Step 4. If $v < SC$, then go to Step 5 otherwise set $i = i + 1$ and go to Step 3.

Step 5. Select k best points found so far. For each point, select N points randomly around x and determine the graph minimizers using the N+1 points. Perform a local search routine on all graph minimizers found and let the best solution found be x_L. If $f(x_L) < \min\{f(xbest)\}$ the set $x^* \leftarrow x_L$; otherwise set $x^* \leftarrow \arg\min\{f(xbest)\}$.

Step 6. Stop the algorithm and output x^* and $f(x^*)$ as the approximate global minimizer and minimum value of problem (P) respectively.

The RRTOpt algorithm uses the parameters $\Delta, N, g_m, k, r, init_points$ and v to solve problem (P). The parameters N and g_m are used for topographical clustering. The parameters $init_point$, k, r and v are used in the RRTOpt algorithm to solve problem (P). The value $init_point$ represents the number of points in the RRT tree T we must have before implementing the stopping criterion $v \leq SC$. The parameter k is the number of points the algorithm performs topographical clustering on. These parameters are adjusted from problem to problem to yield the best results possible. We note that the greater the value of N and k, the more effort the algorithm requires to find the global minimum value. Also, the smaller the value of v then more points in the RRT tree are added to find the global minimum value. The results of the RRTOpt algorithm are presented in chapter 5.

In our analysis of the RRT, we have concluded that it is more efficient to bias the search toward some goal. When solving (P), we assume that we do not know what the global minimum of the problem is. Hence we modify the RRTOpt algorithm presented in this section by biasing the generation of the random point p. In particular, we modify RRTOpt to bias the search towards the best 3 points found in the tree T. In addition, we set a paramter $init_points$ to be the number of points to add to the RRT tree before biasing towards the best 3 points begins. The modification is presented in the next section.

4.4 The RRTOptv1 algorithm

RRTOptv1 omits topographical clustering and performs a local search routine periodically. The overview of it is given below. At every iteration the RRTOptv1 algorithm,

- selects a point $p \in X \cap Z^n$ to find a point x to add the vertex of the RRT tree.
- performs local search on the best z points and save the local minimizers in a array $xbest$, after every l iterations.
- uses the best local minimizers as biasing points, once the iteration counter has reached the value $init_points$. In every subsequent iteration, the random point, p, selected will

be biased toward these 3 best points in the RRT (3 best minimizers) in $xbest$ with some probability.

- updates the z best local minimizers in the matrix $xbest$, if necessary, and keeps iterating until the stopping criterion $v < SC$ is met.

In this algorithm, local search is performed after every l iterations. The local minimizers from the best z points are saved and the iteration continues until $i = i + l$. At this stage, the local minimization is performed on the best z starting from $i + 1$ to $i + l$. The process is continued until the stopping criterion is met.

To demonstrate the effectiveness of the RRTOptv1 we use Figure 4.4.

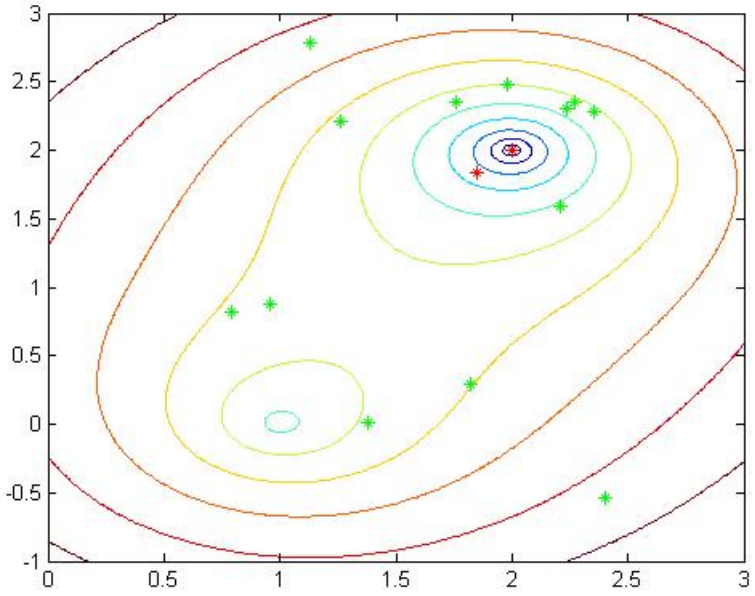


Figure 4.4: The effect of biasing

We generated three points randomly, denoted by the array $\{xbest1, xbest2, xbest3\}$, from the search space. The subsequent points were selected in the following manner:

- Select a random point p^r.

- Generate a biased point p using the equation

$$p = p^r + (xbest1 - p^r) * p_1 + (xbest2 - p^r) * p_2 + (xbest3 - p^r) * p_3, \tag{4.1}$$

  where (p_1, p_2, p_3) is a probability vector.

- Round p and add it to the RRT tree. If the function value of p is better than any point in the array $\{xbest1, xbest2, xbest3\}$, it replaces the worst point in that array. Otherwise, swop p_1 and p_2 in the following iteration when using equation (4.1).

Using the above mentioned procedure of adding points to the RRT tree creates points that are biased closer towards the best point found thus far. This is depicted in Figure 4.4 where most of the points are clustered around the two minimizers.

The points in the array $\{xbest1, xbest2, xbest3\}$ may be comprised of:

1. the three best local minimizers found,
2. the three best points found in the RRT tree or
3. a combination of the best local minimizers and best points found in the RRT tree.

It is a worth experimenting with these three possibilities when implementing the RRTOptv1 algorithm. The results using these the three variations will be presented in the following chapter.

Before we begin the step by step description of the RRTOptv1 algorithm, we describe the parameters. The parameter *init_points* indicates how many points must be in the RRT tree T in order for biasing towards the 3 best point (3 best local minimizers) begins. The parameters z and l represent how many points must be used for local search and after how many iterations must these local searches begin respectively. SC represents the value which the average of the last six Δs (v) must be greater than or equal to for the RRTOptv1 algorithm to stop. We now present a step by step description of the RRTOptv1 algorithm.

Algorithm 8: The RRTOptv1 algorithm

Step 1. Set parameters $init_points, \Delta, l$ and z. Initialize counters $i = 1$ and v to zero, and input the value SC.

Step 2. Select randomly a point $p \in X \cap Z^n$ and set this as the root of the RRT by calling the function BUILD_RRT(P).

Step 3. Select randomly a point $p \in X \cap Z^n$ and the extend the RRT tree using SELECT_EXTENSION(T) function to add the point x to the RRT tree. Update $i = i + 1$.

3.1 If $i <= init_points$ and $gcd(i, i + l) = l$, then perform local search at each of the best z points and save the local minimizers in $xbest$. Go to Step 4.

3.2 The random selection of the point p is biased towards $xbest$ with some probability using probability vector (p_1, p_2, p_3) using equation (4.1).

3.2.1 If $gcd(i, i + l) = l$ then perform local search at each of the best z points found in the previous l iterations. Save the local minimizers and update $xbest$ if necessary. Go to Step 4.

3.2.1 Otherwise if the function value of random point p is less than any of the best 3 local minimizers in $xbest$, then replace the $xbest3$ with p in the $xbest$ array. Go to Step 4.

Step 4. If $v < SC$, then go to Step 3. Otherwise go to Step 5.

Step 5. Stop the algorithm and output $x^* = \arg\min\{f(xbest)\}$ and $f(xbest)$ as the approximate global minimizer and minimum value of problem (P) respectively.

The RRTOptv1 algorithm uses parameters $init_points$, SC, Δ, z and l to solve problem (P). The RRTOpt algorithm was modified in this way to increase the probability of finding the global minimizer x^* of problem (P). If we update the matrix of the 3 best points found so far ($xbest$), then the biasing takes effect on the best *possible* points found thus far. Results of this algorithm tested on the 20 benchmark problems in NIP are presented in chapter 5.

Chapter 5

Numerical results

In this chapter, we present the computational results of the proposed MBLS and RRT-based algorithms. The benchmark test cases contains 8 unconstrained problems and 12 constrained problems. For the constrained problems we considered a penalty function approach ($F(x;c) = f(x) + c\phi(x)$) where the penalty parameter c (see Chapter 2.3.2) in [2, 6] has been used as suggested in [9]. We have done this in our implementation of all the algorithms for a fair comparison. All tests were performed on a personal computer with CPU Pentium 2.39 GHz and 1.95 GB of RAM. All the algorithms were programmed using MATLAB 7.3. In the first section, we present the results of MBLS and justify some of its effectiveness as a local search algorithm. In the second section, we present the results of the topographical clustering and show its strength in nonlinear integer programming. Finally, our newly developed RRTOpt and RRTOptv1 algorithms are compared with recently developed algorithms given in [2, 6, 28] for nonlinear integer programming in Sections 5.3 and 5.4 respectively.

5.1 Results of MBLS

To show the effectiveness of the local search MBLS, we tested it on problem 5 listed in the Appendix. We use ten random starting points, i.e. x_0, and calculated the corresponding x_L (the local minimizer). We present the results in Table 5.1.

In Table 5.1, columns 1 and 2 respectively contain the random starting points and the function values at these points; column 3 contains the number of descent directions of $q(x)$ from

x_0, i.e. $|\{\underline{d}^1, \underline{d}^2, \cdots, \underline{d}^p\}|$; column 4 presents the direction vector d^i at x_0 that lead to x_L. If the non-descent direction vector at x_0 produces x_L then this has been indicated by a '0' in the column 5. The entry '1' indicates that a descent direction of $q(x)$, at x_0, produces x_L of $f(x)$. The number of function calls required to reach x_L is presented in the last column vector under 'f-call'.

Table 5.1: Performance of MBLS on Problem 5, $n = 5$.

x_0	$f(x_0)$	# d^j: $d^{(j)\mathbf{T}}\nabla q(x_0) < 0$	Best d^i	Success	x_L	$f(x_L)$	f-call
(1,-4,-1,-5,-1)	3307	7	(0,1,-1,1,1)	1	(1,-2,-3,-3,1)	175	38
(3,1,2,4,3)	628	7	(0,1,0,-1,0)	1	(3,3,2,2,3)	108	30
(0,4,0,3,0)	2037	6	(0,-1,1,-1,0)	1	(0,2,2,1,0)	85	36
(3,0,-4,-3,-5)	1486	8	(0,-1,0,0,-1)	1	(3,-3,-4,-3,-5)	541	35
(-1,2,-2,1,-3)	138	5	(0,0,0,0,1)	1	(-1,2,-2,1,1)	58	35
(4,0,-3,-3,3)	1280	7	(0,-1,0,0,0)	1	(4,-4,-3,-3,3)	160	51
(1,2,2,-4,5)	421	9	(1,0,1,0,0)	0	(2,2,3,-4,5)	271	40
(-1,5,0,1,-5)	4797	10	(-1,0,1,-1,0)	1	(-5,5,5,-5,-5)	72	28
(0,-5,-1,-1,-2)	4247	7	(0,1,0,0,0)	1	(0,0,-1,-1,-2)	22	31
(3,4,-2,-3,-4)	756	6	(0,-1,0,0,0)	0	(3,3,-2,-3,-4)	203	32

As seen in Table 5.1, the quadratic approximation in the MBLS algorithm, for this particular problem, has a success rate[1] of 80% (there are eight successes in the column under 'Success' out of 10 independent runs). That is, we were able to obtain a local minimizer by taking successive steps in a particular *descent* direction $\underline{d}^i$. We also see in column 4 that the directions d^i, that have lead to a local minimizer, x_L, are not neccessarily unit coordinate vectors (as is the case when using DLS). This is due to the flexiblity of the MBLS algorithm, see Chapter 2.3.1.

These results, in some sense, justify the use of a quadratic approximation in a local search technique. But one has to note that this is very much a function-dependent method. If $f(x)$ has plateau in many regions then one can just resort to using DLS instead (see Chapter 2.2). We now present the results obtained by using multi-start and topographical clustering technique

[1] We ran the MBLS algorithm 100 times with 100 starting points (for the same problem) and found that the success rate, of the quadratic approximation in the MBLS algorithm, is 78%.

together with two different local search procedures.

5.2 Results of the topographical clustering

In this section, we present the justification of the use of topographical clustering to aid in solving problem (P). Given the framework of multi-start (MS) (see Chapter 2.4), any local search can be incorporated with it. Hence, to see the effectiveness of the MS with local search, we have used three local searches with MS. In each instance we do not only look at the approximate global minimizer x^* obtained by the respective methods, but we also look at the median function values of each method given by x_{med} and the maximum function evaluation $x_{\max}$. To give a fair account of the investigation, all four implementations of MS are stopped after the same number of function calls.

Table 5.2 shows the resulfts of this experiment. The second column in Table 5.2, labeled 'MS + DLS', shows the results of the multi-start and DLS used in conjunction; column three shows the results of multi-start embedded with our own MBLSr (see Chapter 2.3.3); the third column shows the results of multi-start combined together with topographical clustering and DLS; and the last column shows the results of multi-start combined with topographical clustering and MBLSr. Each row in the table contains the approximate global minimizer x^*, the median point x_{med} and the maximum point x_{max} that have been found by each method. The function evaluations' limit that we have set for test problems 4, 6, 7, 10 and 11 are 1000, 200000, 35000, 1500000, 15000000 respectively. The parameters of the topographical clustering method are as follows: $N = 12$ and $g_m = 3$.

Table 5.2 shows that MS combined with MBLSr, MS combined with DLS and topographical clustering combined with MBLSr failed to find the correct global minimizer once. The only method which was successful in finding the correct global minimum for all these test problems was topographical clustering combined with DLS. If we look at the median function values of all test problems in Table 5.2, we see that the topographical clustering method (combined with either local search routines) is generally superior than MS combine with DLS. For instance, if we look at problem 6 we see that both the median x_{med} function values for the topographical clustering method is lower than that of MS combined with any local search. This is a direct

result of performing local search on only the graph minimizers of each problem. We see that for all the test problems, the points x_{med} that correspond to the topographical clustering method, have lower function values. This implies that topographical clustering is better equipped to find to global minimum of the problem (P).

One can conclude from this observation that, in general, DLS combined with topographical clustering is superior to all the other methods. Topographical clustering combined with DLS seems to have an edge over topographical clustering combined with MBLSr. Hence, MBLS is not embedded in the newly proposed RRT-based optimizers. With this consideration, we now present the results of the RRTOpt algorithm in the following section.

5.3 Results of the RRT-Optimizer

In this section, we present the results of the RRTOpt algorithm. Since global efficiency is usually defined as the effort the algorithm needs to be successful [2], we record the number of function calls to reach a global minimizer. These results are summarized in Table 5.3. The key parameters used in the RRTOpt algorithm can be found in Table B.1 (see Appendix B). Full descriptions of these parameters are given in Section 4.3 (see page 51). We have compared the results obtained by the RRTOpt algorithm with the results by the discrete dynamic convexized (DC) method [2, 6]. We have used the results prescibed in [2, 6]. Furthermore, we compared the results of the RRTOpt algorithm to those of a discrete filled function method by Ng and Zhang [28]. These results are taken from [28].

Upon attempting to implement the DC method [2, 6] we encountered a few problems. If we look at Step 1 of the formal DS algorithm (see Chapter 3.2.2), the parameter N_L was not prescribed by the author in his presentation of results found in [2]. Also, upon our attempted implementation, we encountered a problem in Step 4, particularly the case $x' \neq x_0$ and $f(x') > f(x^*)$. If k is increased by δ_k, y is assigned x' and Step 4 is repeated, then there seems to be an infinite loop that occurs. Due to this flaw, we were not able to verify the results presented in [2] and [6].

In Table 5.3, every number in the column 'min' is the minimal number of function calls to reach a discrete global minimizer among twenty-five runs of the RRTOpt algorithm; every

Table 5.2: Comparison of methods

Problem	MS + DLS	MS + MBLSr	Top + DLS	Top + MBLSr
4	$x^* = [50 \quad 25 \quad 1.5 \quad 0]$ $x_{med} = [14 \quad 6 \quad 3.5 \quad 0.0285]$ $x_{\max} = [2 \quad 2 \quad 3.5 \quad 0.0285]$	$x^* = [53 \quad 24 \quad 3 \quad 0.0000063]$ $x_{\max} = [55 \quad 7 \quad 4.5 \quad 0.0285]$ $x_{med} = [89 \quad 0 \quad 1 \quad 2.1994]$	$x^* = [50 \quad 25 \quad 1.5 \quad 0]$ $x_{med} = [80 \quad 15 \quad 1.5 \quad 0.0002955]$ $x_{\max} = [100 \quad 25 \quad 2 \quad 0.02747]$	$x^* = [56 \quad 23 \quad 1.5 \quad 0.000022]$ $x_{med} = [66 \quad 20 \quad 1.5 \quad 0.0001847]$ $x_{\max} = [82 \quad 7 \quad 3 \quad 0.009999]$
6	$x^* = [0 \quad -1 \quad 3]$ $x_{med} = [1 \quad -2 \quad 5278]$ $x_{\max} = [-2 \quad 1 \quad 13603]$	$x^* = [0 \quad -1 \quad 3]$ $x_{med} = [2 \quad 1 \quad 2275]$ $x_{\max} = [2 \quad -2 \quad 316600]$	$x^* = [0 \quad -1 \quad 3]$ $x_{med} = [-1 \quad 0 \quad 278]$ $x_{\max} = [-1 \quad 0 \quad 278]$	$x^* = [0 \quad -1 \quad 3]$ $x_{med} = [2 \quad 0 \quad 1736]$ $x_{\max} = [2 \quad 0 \quad 1736]$
7	$x^* = [15 \quad 3 \quad 6 \quad 2.9627]$ $x_{med} = [4 \quad 6 \quad 14 \quad 8.7782]$ $x_{\max} = [1 \quad 1 \quad 22 \quad 35.2600]$	$x^* = [16 \quad 4 \quad 4 \quad 2.8175]$ $x_{med} = [4 \quad 6 \quad 14 \quad 8.7782]$ $x_{\max} = [9 \quad 1 \quad 27 \quad 460.2449]$	$x^* = [16 \quad 4 \quad 4 \quad 2.8175]$ $x_{med} = [8 \quad 6 \quad 10 \quad 4.5986]$ $x_{\max} = [2 \quad 8 \quad 14 \quad 17.1565]$	$x^* = [16 \quad 4 \quad 4 \quad 2.8175]$ $x_{med} = [11 \quad 8 \quad 5 \quad 3.5266]$ $x_{\max} = [6 \quad 14 \quad 4 \quad 6.0758]$
10	$x^* = [3 \quad 0.5 \quad 0]$ $x_{med} = [0 \quad -10 \quad 14.2031]$ $x_{\max} = [0 \quad 6 \quad 14.2031]$	$x^* = [3 \quad 0.5 \quad 0]$ $x_{med} = [0 \quad 4.5 \quad 14.2031]$ $x_{\max} = [10 \quad -8 \quad 26697546.7]$	$x^* = [3 \quad 0.5 \quad 0]$ $x_{med} = [0 \quad 8 \quad 14.2031]$ $x_{\max} = [0 \quad -9.5 \quad 14.2031]$	$x^* = [3 \quad 0.5 \quad 0]$ $x_{med} = [2 \quad -0.5 \quad 2.9532125]$ $x_{\max} = [0 \quad 9.5 \quad 14.2031]$
11	$x^* = [1 \quad 1 \quad 0]$ $x_{med} = [2 \quad 4 \quad 1]$ $x_{\max} = [3 \quad 9 \quad 4]$	$x^* = [1 \quad 1 \quad 0]$ $x_{med} = [0 \quad 6 \quad 3601]$ $x_{\max} = [10 \quad 2 \quad 961914.3]$	$x^* = [1 \quad 1 \quad 0]$ $x_{med} = [2 \quad 4 \quad 1]$ $x_{\max} = [3 \quad 9 \quad 4]$	$x^* = [1 \quad 1 \quad 0]$ $x_{med} = [3 \quad 9 \quad 4]$ $x_{\max} = [2 \quad 5 \quad 101]$

Table 5.3: Performances and comparisons of algorithms on problems

Problem	Our algorithm's results				Results by [2]		Results by [6]		Results by [28]
	min	max	med	fail	med	fail	med	fail	
1	52900	76453	71749	1	24679.2	0	60360.6	0	4474
2	33276	46231	38357	0	13655.1	0	-	-	621
3	255	973	405.2	0	658.1	0	599.2	0	7887
4	1876	4271	2927.1	1	689.7	0	-	-	1574.2
5	2713	3428	3070.6	0	752.8	0	-	-	-
6	5235	9268	6170.1	0	55685.9	0	-	-	-
7	4922	8181	6336.1	2	7055.2	0	11568.7	0	-
8	70658	70878	70762.7	1	77112.8	0	-	0	-
9	30500	78061	69065.7	0	52000.9	0	31162.1	0	-
10	143498	354638	239731.7	0	756941.2	0	-	-	607880.7
11	355102	1004754	642435.9	0	12780839	0	10982038.6	0	1608067.3
12($n = 25$)	3361	47321	37444.8	0	14485	0	-	-	102883.2
12($n = 50$)	65389	89556	83889.2	0	69510	0	-	-	9840.7
12($n = 100$)	350970	421162	385348.5	0	273840	0	-	-	6704633
12($n = 150$)	482852	532611	518149	0	373620	0	-	-	-
13($n = 25$)	22470	39844	35582.7	1	6530	0	-	-	90568.1
13($n = 50$)	97703	138651	125535.2	1	24740	0	-	-	727641.2
13($n = 100$)	122464	209732	179627.7	2	88640	0	-	-	5861265.4
13($n = 150$)	409142	644734	447452.9	2	235840	0	-	-	-
13($n = 200$)	873931	1230024	995140	2	373620	0	-	-	-
14($n = 25$)	22518	31204	30256	0	19675	0	-	-	116765.5
14($n = 50$)	105727	183214	168823	0	157900	0	-	-	997241.3
14($n = 100$)	931834	1682374	1114609	0	1203480	0	-	-	7770218.8
15($n = 25$)	22803	30201	27481	0	28089.6	0	25553.9	0	45158.5
15($n = 50$)	65814	104521	86982.8	0	180368.3	0	204070.5	0	323156.5
15($n = 100$)	526807	946718	730622.9	1	1559704	0	1249832.2	0	2734844.5
15($n = 200$)	988167	1978235	1780499	1	10234584	0	-	-	22585087.68
16($n = 4$)	33952	374264	173388	0	16622286	0	-	-	5105399.5
17 ($n = 25$)	5608	46416	15574.8	0	44050	0	-	-	-
17($n = 50$)	24891	141988	52995.5	0	161600	0	-	-	-
17($n = 100$)	131604	643069	243487	0	567984	0	-	-	-
18($n = 25$)	871196	1799556	1002918	1	538368	0	-	-	-
18($n = 50$)	651063	2233046	1904322.6	1	873828	0	-	-	-
18($n = 100$)	1048398	25044019	21423783.7	2	594520	0	-	-	-
19 ($n = 25$)	27173	64003	43018	0	17878	0	-	-	-
19 ($n = 50$)	73780	816628	139391	1	72640	0	-	-	-
19 ($n = 100$)	309743	536843	447804.7	1	255312	0	-	-	-
20 ($n = 25$)	1340	18462	8673	0	5146	0	-	-	-
20 ($n = 50$)	6139	49268	22657.1	0	19960	0	-	-	-
20 ($n = 100$)	8123	54923	23640.6	0	85200	0	-	-	-

number in the column 'max' is the maximal number of function calls to reach a discrete global minimizer among the runs of the RRTOpt algorithm; every number in the column 'med' is the average number of function calls of twenty-five runs of the RRTOpt algorithm (irrespective of failure to reach the global minimizer); every number in the column 'fail' is the number of runs that the optimum has not been reached among 25 runs.

From the 'fail' column, we see that the RRTOpt algorithm is not always 100% successful in solving all twenty test problems. However, it has a success rate of 90 % and above for all of the test problems. It can also be seen that the RRTOpt algorithm out performs the results given by the discrete function filled method found in [28] on problems 10-15. These problems are constrained and unconstrained problems with the search space, X, being very large in size. Eventhough the discrete function filled method was not tested using randomly generated initial points, the RRTOpt algorithm shows a lower average number of function calls.

Our algorithm also compares reasonably well with Zhu's DC algorithm [2]. It outperforms DC in test problems 3, 5, 7, 8, 10, 11, 14, 15 and 20. All these problems are diverse in nature with dimensions ranging from two up to 200 and also being constrained or unconstrained. This shows the versatility of the RRTOpt algorithm. For example, comparing the specific results of problem 11, we see that the average function call for the RRTOpt algorithm is 642435.9 while the DC algorithm could only manage to achieve 12780839. This is a huge difference of 12138403.1 function evaluations. This is attributed to how the RRT searches unexplored regions of the space and the topographical clustering method which searches efficiently around k 'potential' points.

When one observes the parameters used in the RRTOpt algorithm (see Table B.1 in Appendix B), we can arrive at some initial conclusions for this particular algorithm.

1. The parameter *init_points* is proportional to the search space X. This means that the bigger the search space, the more points the algorithm needs to put into the tree before biasing of the search begins.

2. In general, the bigger the dimension of the problem, the larger N must be. This makes sense when considering the topographical clustering method.

3. The parameter Δ needs to increase with the size of the search space X. If Δ is too small,

the search becomes too fine and the number of points generated by the algorithm increases too much and the search becomes exhaustive.

4. The paramter k is very much function dependent. If a function has many plateau regions then this parameter is increased since the global minimal value cannot be easily found.

The results given in this section show the great promise that the RRT-Opt algorithm proposed can compete with recently developed algorithms for nonlinear integer programming. Thus, we can conclude that RRTOpt algorithm is competitive when we compare its results to recently developed algorithms. It also seems to be a very stable[2] in its quest to solve problem (P).

5.4 Results of the RRTOptv1 algorithm

We now present the results of the RRTOptv1 algorithm in Table 5.4 below. There are five parameters for the RRTOpv1 algorithm. The parameters used for this algorithm can be found in Table B.2 in Appendix B. Full description of these parameters can be found in Subsection 4.4. In addition to these parameters, the probability vector $(p_1, p_2, p_3) = [0.6, 0.24, 0.15]$ is used for all the test problems. With regard to the array $\{xbest1, xbest2, xbest3\}$ (See Chapter 4.4) used in equation (4.1), all the points in this array are the best local minimizers found so far. The results when this array contains the best points found in the RRT tree or contains a combination of the best local minimizer together with the two best points in the RRT tree, are tabulated in the Appendix C and Appendix D respecitvely [3]. For the particular case in Appendix D, we used a representative set of problems to see the trend. We did not test the algorithm on all twenty test problems.

Analyzing the results given in Table 5.4, we immediately notice that the RRTOptv1 algorithm converges to the correct solution for all the 20 test problems. This is in star contrast to the RRTOpt algorithm of the previous section. The RRTOptv1 algorithm has a 100% success rate. We suspect that performing z local searches every l iterations and biasing the search towards

[2]If one looks at the columns 'min', 'max' and 'med' in Table 5.3 , we see that the function evaluations of our method do not greatly vary.

[3]The results presented in Appendix C and Appendix D were derived using the table of parameters in Appendix B.

the best possible points in the matrix *xbest* have increased the probability of finding the global minimizer dramatically. Also the parameter values for *init_points* and Δ were changed for most test problems in comparison to the RRTOpt. This was done to assist the RRTOptv1 algorithm to converge to the correct solution. All the paramter values may be found in the Appendix A. The alterations to the RRTOpt algorithm has made the RRTOptv1 algorithm more robust.

The RRTOptv1 has proven to be more efficient than the discrete dynamic convexized method [2] when looking at problems 2, 3, 6-11, 12($n = 50$), 16, 17, 19 and 20[4]. The RRTOptv1 algorithm consistently produced on average lower function evaluations than the result presented in [2]. This can be seen by comparing columns 4 and 6 of Table 5.4. It can also be seen that when the dimension of any problem is increased, the average function evaluations increases. However, the discrete dynamic convexized method uses less average function evaluations on problems 1, 4, 5, 12($n = 25, 100, 150$)-15 and 18. It can be seen that the RRTOptv1 algorithm solves more variety of problems than DC.

Comparing columns 4 and 7 of Table 5.4, the RRTOptv1 has shown some superiority by having lower average function evaluations on problems 7, 9, 11 and 15($n = 100$) when compared to new version of DC (see Algorithm 4 in Chapter 3.2.2) [6]. This algorithm has not yet been shown to be robust since it was only tested on 6 test problems.

When comparing the RRTOptv1 algorithm to the discrete filled function method [28], we observe that our algorithm uses less function evaluations for problems 3, 10, 12, 13, 14, 15 ($n = 100, 200$) and 16. This algorithm could only solve 10 test problems and the results for problems 12($n = 150$), 13($n = 150, 200$) were not provided. However, discrete filled function method obtained lower average function evaluations for the constrained problems 1, 2 and 4. Again, this algorithm does not seem to be robust since it can solve only a limited number of problems.

The DS algorithm presented in [2, 6] reported a 100% success for all the problems. However, in addition to the implementation difficulties we have presented earlier, no discussions on parameter telling were presented by Zhu [2, 6]. It is therefore not clear if these results are based on different values of the same parameter. Under these circumstances, we believe that the results obtained by the RRTOptv1 algorithm are comparable to DC for some problems while for the

[4]For the unconstrained problems, the RRTOptv1 achieved lower function evaluations for all the dimensions.

Table 5.4: Performances and comparisons of algorithms on problems (RRTOptv1)

Problem	Our algorithm's results				Results by [2]		Results by [6]		Results by [28]
	min	max	med	fail	med	fail	med	fail	
1	231605	322142	283280.3	0	24679.2	0	60360.6	0	4474
2	11703	12928	12242	0	13655.1	0	-	-	621
3	480	793	616.2	0	658.1	0	599.2	0	7887
4	1222	2511	1898.1	0	689.7	0	-	-	1574.2
5	1150	3431	2259.4	0	752.8	0	-	-	-
6	9498	18283	12311.1	0	55685.9	0	-	-	-
7	4017	4222	4150.6	0	7055.2	0	11568.7	0	-
8	34637	39974	37448.2	0	77112.8	0	-	0	-
9	23943	31865	26648	0	52000.9	0	31162.1	0	-
10	126006	188018	1619100.4	0	756941.2	0	-	-	607880.7
11	1135463	2202946	1767100	0	12780839	0	10982038.6	0	1608067.3
12($n = 25$)	14285	16456	15506	0	14485	0	-	-	102883.2
12($n = 50$)	57189	66163	63907	0	69510	0	-	-	9840.7
12($n = 100$)	329747	352600	345670	0	273840	0	-	-	6704633
12($n = 150$)	741137	854423	783090	0	373620	0	-	-	-
13($n = 25$)	67792	72173	70010	0	6530	0	-	-	90568.1
13($n = 50$)	175905	194602	184770	0	24740	0	-	-	727641.2
13($n = 100$)	768138	798675	781640	0	88640	0	-	-	5861265.4
13($n = 150$)	1782822	1973133	1869500	0	235840	0	-	-	-
13($n = 200$)	3147465	3280667	3216800.8	0	373620	0	-	-	-
14($n = 25$)	92256	110467	10090	0	19675	0	-	-	116765.5
14($n = 50$)	521101	590530	541120	0	157900	0	-	-	997241.3
14($n = 100$)	1680494	1910344	1797700	0	1203480	0	-	-	7770218.8
15($n = 25$)	107150	190899	143630	0	28089.6	0	25553.9	0	45158.5
15($n = 50$)	403618	490134	454210	0	180368.3	0	204070.5	0	323156.5
15($n = 100$)	1611402	1980104	1835100	0	1559704	0	1249832.2	0	2734844.5
15($n = 200$)	6290512	7571290	6854200.7	0	10234584	0	-	-	22585087.68
16($n = 4$)	1762234	2377452	2003349	0	16622286	0	-	-	5105399.5
17 ($n = 25$)	16251	20054	18269	0	44050	0	-	-	-
17($n = 50$)	71602	84345	71171	0	161600	0	-	-	-
17($n = 100$)	300507	356260	327590	0	567984	0	-	-	-
18($n = 25$)	5933117	9549681	8124051	0	538368	0	-	-	-
18($n = 50$)	29790298	33318157	32219000	0	873828	0	-	-	-
18($n = 100$)	123921720	131421550	127238490.6	0	594520	0	-	-	-
19 ($n = 25$)	19945	23374	21242	0	17878	0	-	-	-
19 ($n = 50$)	92062	103414	98499	0	72640	0	-	-	-
19 ($n = 100$)	200318	218164	208769	0	255312	0	-	-	-
20 ($n = 25$)	3863	4316	4071.2	0	5146	0	-	-	-
20 ($n = 50$)	6516	8605	7511	0	19960	0	-	-	-
20 ($n = 100$)	22082	27056	24141	0	85200	0	-	-	-

other problems RRTOptv1 is superior.

5.5 Summary

Although the MBLS algorithm has shown some promise, it cannot be regarded as a robust local search procedure. It is too dependent on the nature of the landscape of the objective function $f(x)$. If the function has too many plateau regions then in general it is more efficient to use DLS since the effort of performing a quadratic approximation from x_0, the point in question, is wasted. We conclude that it is best, for now, to use DLS as a local search procedure.

A comparison of RRTOpt and RRTOptv1 shows that the RRTOptv1 algorithm is superior. It is also superior to the recent methods [2, 6, 28] due to its iterative nature and its probabilistic choice of the random point p.

Chapter 6

Conclusion

The objective of this dissertation was devoted to design an efficient and robust algorithm for nonlinear integer programming (NIP). In order to achieve this objective, we have proposed an algorithm that uses rapidly-exploring random tree (RRT) as a search space exploration tool rather than the commonly used multi-start. The use of RRT algorithm in the field of global optimization has never been investigated before. We have developed two algorithms. These are RRTOpt and RRTOptv1.

We have done an extensive review of the most recently developed algorithms for NIP. These algorithms use multi-start technique as their search space navigation tool along with an auxiliary function. These were shown to navigate through the search space inefficiently and hence have poor global efficiency.

We have proposed a new local search technique called the model-based local search (MBLS). It was found that this local search gives an indication of which is the most descent direction from the point in question. We attempted to introduce the concept of Voronoi diagrams in our quest to solve the NIP problems but found a major flaw of computing the Voronoi cell sizes and the inefficiency of computing a Voronoi diagram at each iteration of the algorithm.

The RRT algorithm was found to be a better search space algorithm provided some bias is introduced. The news algorithms faired well against recently developed algorithms to solve the NIP problems. We have presented extensive numerical results and have shown that the new algorithms have a role to play in solving nonlinear integer programming problems.

A possible extension to this research is to develop theory to calibrate the parameters used

in both the RRTOpt and RRTOptv1 algorithms. Furthermore, one may seek to test different biasing methods in the RRTOptv1 algorithm.

Appendix A

Test problems

In this appendix, we present twenty known problems which are often used by nonlinear integer programming researchers. These problems represent various characteristic terrain found in real-world problems,.e.g., unimodal or multimodal, with or without plateaus and ridges, and high or low dimensional. Some of these test problems can be found in textbooks, in individual research articles, or at different websites. Please note that in several cases, the global minimizer x^* and the corresponding global minimum $f(x^*)$ are known only as a numerical approximation.

Problem 1.

$$\begin{cases} \min & x_1^2 + x_2^2 + 3x_3^2 + 4x_4^2 + 2x_5^2 - 8x_1 - 2x_2 - 3x_3 - x_4 - 2x_5, \\ \text{s.t.} & x_1 + 2x_2 + 2x_3 + x_4 + 6x_5 \leq 800 \\ & 2x_1 + x_2 + 6x_3 \leq 200 \\ & x_3 + x_4 + 5x_5 \leq 200 \\ & x_1 + x_2 + x_3 + x_4 \geq 48 \\ & x_2 + x_4 + x_5 \geq 34 \\ & 6x_1 + 7x_5 \geq 104 \\ & 55 \leq x_1 + x_2 + x_3 + x_4 + x_5 \leq 400 \\ & 0 \leq x_i \leq 99, x_i : \text{integer}, i = 1, 2, 3, 4, 5. \end{cases}$$

A discrete global minimizer is $(16, 22, 5, 5, 7)^{\mathrm{T}}$, and the global minimal value is 807. We take a penalty $p(x) = \max\{0, x_1 + 2x_2 + 2x_3 + x_4 + 6x_5 - 800, 2x_1 + x_2 + 6x_3 - 200, x_3 + x_4 + 5x_5 - 200, -x_1 - x_2 - x_3 - x_4 - 48, -x_2 - x_4 - x_5 + 34, -6x_1 - 7x_5 + 104, x_1 + x_2 + x_3 + x_4 + x_5 - 400, 55 - x_1 - x_2 - x_3 - x_4 - x_5\}$, and a penalty parameter $c = 1000$ to convert this problem to an equivalent box constrained nonlinear integer programming problem.

Problem 2.

$$\begin{cases} \max & x_1^2 + x_1x_2 - x_2^2 + x_1x_3 - x_3^2 + 8x_4^2 - 17x_5^2 + 6x_6^3 \\ & \quad + x_4x_5x_6x_7 + x_8^3 + x_9^4 - x_{10}^5 - x_5x_{10} + 18x_3x_6x_7 \\ \text{s.t} & 0 \leq x_i \leq 99, x_i : \text{integer}, i = 1, 2, \cdots, 10. \end{cases}$$

A discrete global minimizer is $(99, 49, 99, 99, 99, 99, 99, 99, 99, 0)^{\mathrm{T}}$, and the global minimal value is 216300719.

Problem 3.

$$\begin{cases} \min & 5u_1+5u_2+5u_3+5u_4-5u_1^2-5u_2^2- \\ & 5u_3^2-5u_4-(v_1+v_2+\cdots+v_9) \\ \text{s.t} & 2u_1+2u_2+v_6+v_7\le 10 \\ & 2u_1+2u_3+v_6+v_8\le 10 \\ & 2u_2+2u_3+v_7+v_8\le 10 \\ & -2u_4-v_1+v_6\le 10 \\ & -2v_2-v_3+v_7\le 0 \\ & -2v_4-v_5+v_8\le 0 \\ & -8u_i+v_{i+5}\le 0 \quad i=1,2,3 \\ & u_j,v_k\in 0,1, \quad j=1,2,3,4,k=1,2,3,4,5,9 \\ & v_k\in 0,1,2,3, \quad k=6,7,8. \end{cases}$$

A discrete global minimizer is $(u_1,u_2,u_3,u_4,1,1,1,1,3,3,3,1)^{\mathrm{T}}$, for all $u_i\in\{0,1\}$,$i=1,2,3,4$, and the global minimal value is -15. We take a penalty $p(x)=\max\{0,2u_1+2u_2+v_6+v_7-10,2u_1+2u_3+v_6+v_8-10,$ $2u_2+2u_3+v_7+v_8-10,-2u_4-v_1+v_6-10,-2v_2-v_3+v_7,-2v_4-v_5+v_8,-8u_1+v_6,-8u_2+v_7,-8u_3+v_8\}$, and a penalty parameter $c=1000$ to convert this problem to an equivalent box constrained nonlinear integer programming problem.

Problem 4.

$$\begin{cases} \min & \displaystyle\sum_{i=1}^{9}\left(\exp^{-\frac{(u_i-x_2)^{x_3}}{x_1}}-\frac{i}{100}\right)^2 \\ \text{s.t.} & 1\le x_1\le 100, 0\le x_2\le 25 \\ & x_i:\text{integer}, i=1,2. \\ & x_3=\dfrac{j}{2}, 0\le j\le 10j:\text{integer} \\ & u_i=25+\left(-50\dot{\log}\dfrac{i}{100}\right)^{\frac{2}{3}} \end{cases}$$

This problem is a discrete counterpart of Problem 1 in [26]. A discrete global minimizer is $(50, 25, 1.5)^{\mathrm{T}}$, and the global minimal value is approximately 0.0.

Problem 5.

$$\begin{cases} \min & 100(x_2 - x_1^2) + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 \\ & + 10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1) \\ \text{s.t.} & -10 \le x_i \le 10, \text{integer}, i = 1, 2, 3, 4. \end{cases}$$

A discrete global minimizer is $(1, 1, 1, 1)^{\mathrm{T}}$, and the global minimal value is 0.0.

Problem 6.

$$\begin{cases} \min & [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \\ & \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \\ \text{s.t.} & x_i = 0.001j, -2000 \le j \le 2000, j : \text{integer}. \end{cases}$$

A discrete global minimizer is $(0, -1)^{\mathrm{T}}$, and the global minimal value is 3.

Problem 7.

$$\begin{cases} \min & \dfrac{33.7539}{x_1} + \dfrac{1.4430}{x_2} + \dfrac{1.3885}{x_3} \\ \text{s.t.} & x_1 + x_2 + x_3 = 24 \\ & 1 \le x_1 \le 16 \\ & 1 \le x_2 \le 20 \\ & 1 \le x_3 \le 28 \\ & x_i : \text{integer}, i = 1, 2, 3. \end{cases}$$

A discrete global minimizer is $(16, 4, 4)^{\mathrm{T}}$, and the global minimal value of this problem is 2.817494. We take a penalty $p(x) = \|x_1 + x_2 + x_3 - 24\|$ and the penalty parameter $c = 35$ to convert this problem to an equivalent

box constrained nonlinear integer programming problem.

Problem 8.

$$\begin{cases} \min & -x_3 - x_4 - x_5 \\ \text{s.t.} & 20x_1 + 30x_2 + x_3 + 2x_4 + 2x_5 \leq 180 \\ & 30x_1 + 20x_2 + 2x_3 + x_4 + 2x_5 \leq 150 \\ & -60x_1 + x_3 \leq 0 \\ & -75x_2 + x_4 \leq 0 \\ & 0 \leq x_i \leq 1, i = 1, 2 \\ & 0 \leq x_i \leq 75, i = 3, 4, 5 \\ & x_i : \text{integer}, i = 1, 2, 3, 4, 5. \end{cases}$$

A discrete global minimizer is $(1, 1, 24, 52, 0)^{\mathrm{T}}$, and the global minimal value of this problem is -76. We take a penalty $p(x) = \max\{0, 20x_1 + 30x_2 + x_3 + 2x_4 + 2x_5 - 180, 30x_1 + 20x_2 + 2x_3 + x_4 + 2x_5 - 150, -60x_1 + x_3, -75x_2 + x_4\}$ and the penalty parameter $c = 200$ to convert this problem to an equivalent box constrained nonlinear integer programming problem.

Problem 9.

$$\begin{cases} \min & x_1x_2x_3 + x_1x_4x_5 + x_2x_4x_6 + x_6x_7x_8 + x_2x_5x_7 \\ \text{s.t.} & 2x_1 + 2x_4 + 8x_8 \geq 12 \\ & 11x_1 + 7x_4 + 13x_6 \geq 41 \\ & 6x_2 + 9x_4x_6 + 5x_7 \geq 60 \\ & 3x_2 + 5x_5 + 7x_8 \geq 42 \\ & 6x_2x_7 + 9x_3 + 5x_5 \geq 53 \\ & 4x_3x_7 + x_5 \geq 13 \\ & 2x_1 + 4x_2 + 7x_4 + 3x_5 + x_7 \leq 69 \\ & 9x_1x_8 + 6x_3x_5 + 4x_3x_7 \leq 47 \\ & 12x_2 + 8x_2x_8 + 2x_3x_6 \leq 73 \\ & x_3 + 4x_5 + 2x_6 + 9x_8 \leq 31 \\ & 0 \leq x_i \leq 7, i = 3, 4, 6, 8 \\ & 0 \leq x_i \leq 15, i = 2, 5, 7 \\ & x_i : \text{integer}, i = 1, 2, \cdots, 8. \end{cases}$$

A discrete global minimizer is $(5, 4, 1, 1, 6, 3, 2, 0)^{\mathrm{T}}$, and the global minimal value of this problem is 110. We take a penalty $p(x) = \max\{0, -2x_1 - 2x_4 - 8x_8 + 12, -11x_1 - 7x_4 - 13x_6 + 41, -6x_2 - 9x_4x_6 - 5x_7 + 60, -3x_2 - 5x_5 - 7x_8 + 42, -6x_2x_7 - 9x_3 - 5x_5 + 53, -4x_3x_7 - x_5 + 13, 2x_1 + 4x_2 + 7x_4 + 3x_5 + x_7 - 69, 9x_1x_8 + 6x_3x_5 + 4x_3x_7 - 47, 12x_2 + 8x_2x_8 + 2x_3x_6 - 73, x_3 + 4x_5 + 2x_6 + 9x_8 - 31\}$ and the penalty parameter $c = 200$ to convert this problem to an equivalent box constrained nonlinear integer programming problem.

Problem 10.

$$\begin{cases} \min & [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2 \\ \text{s.t.} & x_i = 0.001j, -10^4 \leq j \leq 10^4, j : \text{integer}, \quad i = 1, 2. \end{cases}$$

A discrete global minimizer is $(3, 0.5)^{\mathrm{T}}$, and the global minimal value is 0.0.

Problem 11.

$$\begin{cases} \min & f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ \text{s.t.} & x_1^2 + x_2^2 \geq 0.25 \\ & -\dfrac{1}{3}x_1 + x_2 \geq 0.1 \\ & x_i = j_i \times 10^{-4} \\ & 0 \leq j_i \leq 10^5, j_i : \text{integer}, \quad i = 1, 2. \end{cases}$$

A discrete global minimizer is $(1, 1)^{\mathrm{T}}$, and the global minimal value is 0. We take a penalty $p(x) = \max\left\{0, 0.25 - x_1^2 - x_2^2, 0.1 + \frac{1}{3}x_1 - x_2\right\}$ and the penalty parameter $c = 1000$ to convert this problem to an equivalent box constrained nonlinear integer programming problem.

Problem 12.

$$\begin{cases} \min & (x_1 - 1)^2 + (x_n - 1)^2 + n\sum_{i=1}^{n}(n - i)(x_i^2 - x_{i+1})^2 \\ \text{s.t.} & -5 \leq x_i \leq 5, x_i : \text{integer}, \quad i = 1, 2, \cdots, n. \end{cases}$$

A discrete global minimizer is $(1, 1, \cdots, 1)^{\mathrm{T}}$, and the global minimal value is 0.0.

Problem 13.

$$\begin{cases} \min & \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \\ \text{s.t.} & -5 \leq x_i \leq 5, x_i : \text{integer}, \quad i = 1, 2, \cdots, n. \end{cases}$$

A discrete global minimizer is $(1, 1, \cdots, 1)^{\mathrm{T}}$, and the global minimal value is 0.0.

Problem 14.

$$\begin{cases} \min & \sum_{i=1}^{n} x_i^4 + \left(\sum_{i=1}^{n} x_i\right)^2 \\ \text{s.t.} & -5 \le x_i \le 5, x_i : \text{integer}, \quad i = 1, 2, \cdots, n. \end{cases}$$

A discrete global minimizer is $(0, 0, \cdots, 0)^{\mathrm{T}}$, and the global minimal value is 0.0.

Problem 15.

$$\begin{cases} \min & f(x) = x^{\mathrm{T}} Q x \\ \text{s.t.} & \sum_{i=1}^{n} \dfrac{x_i^2}{9n+i} \le 1 \\ & \sum_{i=1}^{n} i x_i \ge \dfrac{n}{2} \\ & -5 \le x_i \le 5, x_i : \text{integer}, \quad i = 1, 2, \cdots, n. \\ & Q = [Q_{ij}], Q_{i,j} = 1 \quad \text{for } i \ne j. \end{cases}$$

The global minimal value of this problem is 2. We take penalty $p(x) = \max\left\{0, \sum_{i=1}^{n} \frac{x_i^2}{9n+i} - 1, \frac{n}{2} - \sum_{i=1}^{n} i x_i\right\}$, and penalty parameter $c = 10000$ to convert this problem to an equivalent box constrained nonlinear integer programming problem.

Problem 16.

$$\begin{cases} \min & f(x) = (x_1 + 10x_1)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \\ \text{s.t.} & x_i = 0.001j, -10^4 \le x_i \le 10^4, j : \text{integer}, \ i = 1, 2, 3, 4. \end{cases}$$

The only discrete global minimizer is $(0, 0, 0, 0)^{\mathrm{T}}$, and the global minimal value is 0.0.

Problem 17.

$$\begin{cases} \min \quad f(x) = -20\exp\left(-0.02\sqrt{\dfrac{1}{n}\sum_{i=1}^{n} x_i^2}\right) + \exp\left(\dfrac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \\ \text{s.t.} \quad -30 \le x_i \le 30, x_i : \text{integer}, \ i = 1, 2, \cdots, n. \end{cases}$$

This problem is a discrete counterpart of Ackley's problem. The number of discrete local minima is not known. The global minimum is located at the origin with the global minimal value 0.0.

Problem 18.

$$\begin{cases} \min \quad f(x) = 1 + \dfrac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\dfrac{x_i}{\sqrt{i}}\right) \\ \text{s.t.} \quad -600 \le x_i \le 600, x_i : \text{integer}, \ i = 1, 2, \cdots, n. \end{cases}$$

This problem is a discrete counterpart of Griewank's problem. The number of discrete local minima is not known. The global minimum is located at the origin with the global minimal value 0.0.

Problem 19.

$$\begin{cases} \min \quad f(x) = \dfrac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} \\ \text{s.t.} \quad y_i = 1 + \dfrac{1}{4}(x_i + 1), i = 1, 2, \cdots, n \\ \qquad -10 \le x_i \le 10, x_i : \text{integer}, \ i = 1, 2, \cdots, n. \end{cases}$$

This problem is a discrete counterpart of Levy and Montalvo's problem . The number of discrete local minima is not known. The global minimum is located at $(-1, -1, -1, \cdots, -1)^{\mathrm{T}}$ with the global minimal value 0.0.

Problem 20.

$$\begin{cases} \min \quad f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)] \\ \text{s.t.} \quad -5 \leq x_i \leq 5, x_i : \text{integer}, \ i = 1, 2, \cdots, n. \end{cases}$$

This problem is a discrete counterpart of Rastrigin's problem. The number of discrete local minima is not known. The global minimum is located at the origin with the global minimal value 0.0.

Appendix B

Tables of paramters

Problem	N	g_m	Δ	SC	$init_points$	k	r
1	30	10	88	22	12000	60	5
1	30	10	88	22	12000	60	5
3	11	3	2.4	1.8	50	1	5
4	20	7	41	1.5	1300	5	5
5	30	7	16	4	1000	5	5
6	11	3	2262	1000	2300	5	5
7	11	3	14	3.5	2300	5	5
8	11	3	45	12	25000	10	5
9	11	3	12	3	30000	50	5
10	11	3	1300	28	25000	30	5
11	11	3	2500	800	25000	30	5
12	100-300	50	25	10	5000-9000	3-5	5
13	100-300	50	25	13	5000-7000	3	5
14	100	20	25	13	5000-6000	3	5
15	100	20	25	13	5000-8000	3	5
16	11	3	3000	800	5000	2	5
17	11	5	25	12	1000-8000	3	5
18	100-300	50	2000	700	10000-30000	50	5
19	11	3	30	12	1000-3000	2	5
20	11	3	20	12	100-500	2	5

Table B.1: Parameters for RRTOpt

Problem	*init_points*	*SC*	Δ	*l*	*z*
1	900	110.6	11.068	15	3
2	300	156.5	31.3	10	3
3	50	3.041	1.2165	15	1
4	100	51.29	1.025	10	2
5	500	20	8	40	5
6	100	2828.42	1131.37	20	3
7	2000	18.131	7.25	30	3
8	5000	64.9	1.29	50	4
9	600	15.1	1.02	15	2
10	5000	1414.2	5656.8	500	3
11	5000	70710.6	28284.2	500	3
12	800-1250	25-62	10-25	200-250	3
13	600-11000	25-70	10-80	70-1000	5
14	700	25-50	3-5	35	4
15	600	25-70	2-70	15-25	3
16	600	20000	4000	60	3
17	50	15-300	90-180	20	1
18	600	3000-6000	150-306	15	1
19	300-450	50-100	30-60	100	2
20	3000	35	35	2999	1

Table B.2: Parameters for RRTOptv1

Appendix C

Table of RRTOptV1 using biasing towards 3 best points found

Table C.1: Performances and comparisons of algorithms on problems

Problem	Our algorithm's results				Results by [2]		Results by [6]		Results by [28]
	min	max	med	fail	med	fail	med	fail	
1	174873	249629	204220	3	24679.2	0	60360.6	0	4474
2	11835	14013	12807	0	13655.1	0	-	-	621
3	313	481	381	0	658.1	0	599.2	0	7887
4	1373	5207	2679.2	5	689.7	0	-	-	1574.2
5	2604	2939	2790.4	0	752.8	0	-	-	-
6	13760	28509	20486	0	55685.9	0	-	-	-
7	5281	9423	7329.4	5	7055.2	0	11568.7	0	-
8	3491	3841	3693.6	0	77112.8	0	-	0	-
9	27335	45982	39321.5	4	52000.9	0	31162.1	0	-
10	109198	235812	183291.7	0	756941.2	0	-	-	607880.7
11	1367223	2229499	1847631.4	0	12780839	0	10982038.6	0	1608067.3
12($n = 25$)	13748	15217	17342.9	1	14485	0	-	-	102883.2
12($n = 50$)	59135	73198	68912.3	1	69510	0	-	-	9840.7
12($n = 100$)	331942	397184	355348.8	0	273840	0	-	-	6704633
12($n = 150$)	772912	837297	803581.4	0	373620	0	-	-	-
13($n = 25$)	45219	65924	53201.6	3	6530	0	-	-	90568.1
13($n = 50$)	169532	200321	174285	2	24740	0	-	-	727641.2
13($n = 100$)	792145	821349	802553.5	2	88640	0	-	-	5861265.4
13($n = 150$)	1878964	3525530	224439.2	2	235840	0	-	-	-
13($n = 200$)	2929037	4709302	3821004.3	4	373620	0	-	-	-
14($n = 25$)	94333	117323	109249.1	0	19675	0	-	-	116765.5
14($n = 50$)	500248	582111	569613	6	157900	0	-	-	997241.3
14($n = 100$)	1773128	187658	2051040	1	1203480	0	-	-	7770218.8
15($n = 25$)	136932	183218	163879.9	0	28089.6	0	25553.9	0	45158.5
15($n = 50$)	403956	510016	485634	0	180368.3	0	204070.5	0	323156.5
15($n = 100$)	1756590	2199605	201976.3	1	1559704	0	1249832.2	0	2734844.5
15($n = 200$)	6401361	8594235	7923741	1	10234584	0	-	-	22585087.68
16($n = 4$)	1540342	2211137	2037721.2	0	16622286	0	-	-	5105399.5
17 ($n = 25$)	15707	24295	19472.2	0	161600	0	-	-	-
17($n = 50$)	74592	88327	83885.3	0	161600	0	-	-	-
17($n = 100$)	313875	326325	319733.2	0	567984	0	-	-	-
18($n = 25$)	653981	9174432	7342218.3	0	538368	0	-	-	-
18($n = 50$)	32774318	38664971	36432997.1	0	873828	0	-	-	-
18($n = 100$)	187398216	274733265	200234775.2	0	594520	0	-	-	-
19 ($n = 25$)	19751	22831	21993.2	0	17878	0	-	-	-
19 ($n = 50$)	102639	108554	106882	0	72640	0	-	-	-
19 ($n = 100$)	198332	208624	203755.7	0	255312	0	-	-	-
20 ($n = 25$)	3936	5821	4129.6	0	5146	0	-	-	-
20 ($n = 50$)	7210	9106	7821	0	19960	0	-	-	-
20 ($n = 100$)	24723	29672	27331.6	0	85200	0	-	-	-

Appendix D

Table of RRTOptV1 using biasing towards a combination of 1 local minimizer and 2 best points in the RRT tree

Table D.1: Performances and comparisons of algorithms on problems

Problem	Our algorithm's results				Results by [2]		Results by [6]		Results by [28]
	min	max	med	fail	med	fail	med	fail	
1	177268	255127	216840.5	4	24679.2	0	60360.6	0	4474
4	1352	2569	1996.5	6	689.7	0	-	-	1574.2
5	404	555	494.1	6	752.8	0	-	-	-
8	4366	13043	7716.3	7	77112.8	0	-	0	-
12($n = 25$)	13818	17193	15068.3	3	14485	0	-	-	102883.2
12($n = 50$)	58994	65842	62826	0	69510	0	-	-	9840.7
12($n = 100$)	299097	362587	334460.1	0	273840	0	-	-	6704633
12($n = 150$)	750245	895732	800707.3	2	373620	0	-	-	-
14($n = 25$)	136753	152202	144476.3	3	19675	0	-	-	116765.5
14($n = 50$)	535854	582028	551363.8	5	157900	0	-	-	997241.3
14($n = 100$)	174882	204997	198231.7	3	1203480	0	-	-	7770218.8
16($n = 4$)	835066	1697589	1325680	0	16622286	0	-	-	5105399.5
19 ($n = 25$)	20058	27445	25008.6	0	17878	0	-	-	-
19 ($n = 50$)	96771	107992	103231.3	0	72640	0	-	-	-
19 ($n = 100$)	204196	2103743	207985.2	0	255312	0	-	-	-

Bibliography

[1] Horst, R., Pardalos, P., Handbook of Global Optimization, Kluwer Academic Publishers, London, 1995.

[2] Zhu, W., Fan, H., A discrete dynamic convexized method for nonlinear integer programming, Journal of Computational and Applied Mathematics, Vol.223, pp. 356-373, 2009.

[3] Ali, M.M., Kaelo P., Improved particle swarm algorithms for global optimization, Applied Mathematics and Computation, Vol.196, pp. 578-593, 2008.

[4] Sugden, S., A class of direct search methods for nonlinear integer programming, Ph.D Thesis, Bond University, 1992.

[5] R. Storn and K. Price, Differential Evolution – A simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization, Vol.11, pp. 341-359, 1997

[6] Zhu, W., Ali, M.M., Discrete dynamic convexized method for nonlinearly constrained nonlinear programming, Computers and Operations Research, Vol.36, pp. 2723-2728, 2009.

[7] Srivastava, V., Fahim, A., A two-phase optimization procedure for integer programming problems, Computers and Mathematics with Applications, Vol.42, pp. 1585-1595, 2001.

[8] Bertsimas, D., Tsitsiklis, J., Simulated Annealing, Statistical Science, Vol.8, pp. 10-15, 1993.

[9] Sinclair, M., An exact penalty function approach for nonlinear integer programming problems, European Journal of Operations Research, Vol.27, pp. 50-56, 1986.

[10] Li, D., Sun, X., Nonlinear Integer Programming, Springer + Business Media, New York, 2006.

[11] Roque, W., Doering, D., Constructing approximate diagrams from digital images of generalized polygons and circular objects, WSG, Feb.3-7, Plezen Czech Republic, 2003.

[12] Ceppi, S., Gatti, N., Patrini, G., Rocco, M., Local search techniques for computing equilibria in two-player general-sum strategic-form games (Extended Abstract), Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010), vand der Hoek, Kaminka, Lesperance, Luck and Sen (eds.), May, 10-14, 2010, Toronto, Canada, pp. 1469-1470.

[13] LaValle, S., Kuffner, J., Rapidly-exploring Random Trees: Progress and prospects, In B. Donald, K.Lynch and D.Rus, editors, Algorithmic and computational robotics: new directions, pp. 293-308, A.K. Peters, Wellesley, MA, 2001.

[14] Sharifzadeh, M., Shahabi, C., Approximate Voronoi cell computation on geometric data streams, Technical report, Computer Science Department, University of Southern California, 2004. No. 04-835.

[15] Urmson, C., Simmons, R., Approaches for heuristically biasing RRT growth. In IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, 2003.

[16] Radtke, P., Wong, T., Sabourin, R., Classification system optimization with multi-objective genetic algorithms, in Proceedings of the l0th International Workshop on Frontiers in Handwriten Recognition (IWFHR 2006). IAPR, 2006, pp. 331-336.

[17] Arya, S., Malamatos, T., Mount, D., Space-efficient approximate Voronoi diagrams, in: Proc. 34th ACM Sympos. Theory Comput., 2002, pp. 721730.

[18] LaValle, S., Rapidly-exploring random trees: A new tool for path planning, TR 98-11, Computer Science Dept., Iowa State University. http://janowiec.cs.iastate.edu/papers/rrt.ps, Oct. 1998.

[19] Aurenhammer, F., Voronoi diagrams- A survey of a fundamental geometric data structure, ACM Computing Surveys (CSUR). 23, 345405.

[20] Törn, A., Viitanen, S., Topographical global optimization using pre-sampled points, Journal of Global Optimization, Vol.5, pp. 267-276, 1994.

[21] Kim, J., Esposito, J., Kumar, V., An RRT-Based algorithm for testing and validating multi-robot controllers, In: RSS, Boston, MA, pp. 249256, 2005.

[22] Ge, R.,A filled function method for finding a global minimizer of a function of several variables, Math.Program, Vol.46, pp. 191-204, 1994.

[23] Michalewicz, Z., Genetic Algorithms + DataStructures = Evolution Programs, Springer-Verlag, Berlin, 1996.

[24] Rinnoy Kan, A.H.G., Timmer, G.T., Stochastic global optimization methods; Part-I: Clustering methods, Mathematical programming, Vol.39, pp. 27-56, 1987.

[25] Wang, W., Shang, Y., Zhang, L., A new T-F Function and algorithm for nonlinear integer programming, The First International Symposium on Optimization and Systems Biology (OSB'07) Beijing, China, August 8-10, 2007.

[26] Weise, T., Global Optimization Algorithms-Theory and Applications 2nd edition, http://www.it-weise.de/.

[27] Zhu, W., An approximate algorithm for nonlinear integer programming, Journal of Applied Mathematics and Computation, Vol. 98, pp. 183-193, 1998.

[28] Ng, C., Zhang, L., A filled function method for discrete global optimization, Computational Optimization and Applications, Vol. 31, pp. 87-115, 2005.

[29] Tian, P., Ma, J., Zhang, D., Non-linear integer programming by Darwin and Boltzmann mixed strategy, European Journal of Operations Research, Vol.105, pp. 224-235, 1998.