ABSTRACT

YU, LU. An Electromagnetism-like Method for Solving Linearly and Quadratically Constrained Optimization Problems. (Under the direction of Dr. Shu-Cherng Fang.)

In this dissertation, a novel population-based global optimization method is studied. Our goal is to redesign the so-called Electromagnetism-like Mechanism (EM) method for solving optimization problems with linear and convex quadratic constraints. The proposed method mimics the behavior of electrically charged particles that are restricted in a feasible region formed by the constraints. The underlying idea of the method is to direct the sample points to some attractive regions of the feasible space for optimization.

Two EM methods (EM-Lin and EM-CQ) have been developed for solving the linearly constrained problems and convex quadratically constrained problems, respectively. In each algorithm, the major steps are designed based on the original EM method to handle different constraints in an efficient manner to find global optimal solutions. The proposed methods have been evaluated using different test problems in the literature and compared with some existing methods. Moreover, the EM-Lin method is used to solve a practical financial planning problem and compared the results with some known methods proposed in the literature.

The computational results show that without using the higher-order information, EM-Lin and EM-CQ converge rapidly (in terms of the number of function evaluations) to the global optimal solutions and produce better results than other known methods in solving problems of a varying degree of difficulty.

An Electromagnetism-like Method for Solving Linearly and Quadratically Constrained Optimization Problems

by
Lu Yu

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Operations Research

Raleigh, North Carolina

2011

APPROVED BY:

Dr. James R. Wilson	Dr. Negash G. Medhin
Dr. Yahya Fathi	Dr. Shu-Cherng Fang Chair of Advisory Committee

DEDICATION

I dedicate this dissertation to my parents, Hua Yu and Juyun Wang, for their love.

BIOGRAPHY

Lu Yu was born in Harbin, China on June 28, 1984, and spent his first 17 years in the city. He entered the School of Mathematical Science of Peking University in 2002 and received his bachelor degree in Mathematical Science in 2006. In August 2006, he joined the Graduate Program in Operations Research of the North Carolina State University. In the Summer of 2008, he received his master's degree in Operations Research.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude and appreciation to my advisor, Dr. Shu-Cherng Fang, for his guidance, encouragement and support in my dissertation research. I would also like to thank other committee members, Dr. Yahya Fathi, Dr. Negash G. Medhin and Dr. James R. Wilson, for their valuable comments and suggestions.

My appreciations also go to Qingwei Jin and Pingke Li for their helpful suggestions. I would like to convey my gratitude to all my fellow graduate students in the Fuzzy and Neural Group (FANGroup) and Operations Research program. Especially, Yu-Min Lin, Pingke Li, Lan Li, Kun Huang, Qingwei Jin, Ye Tian, Yuan Tian, Pu Wang, Chia-Chun Hsu, Zhibin Deng, Ziteng Wang, Cheng Lu, Lihui Zhang, Ling Gai and Tao Hong.

Finally, I would like to thank my family for their love, understanding, encouragement and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

In engineering, economic and scientific studies, quantitative decisions are frequently modeled by optimization concepts and tools. A decision maker may want to find the best decision which corresponds to the minimum (or maximum) of an objective function, while it satisfies a given collection of feasibility constraints.

A large number of such models fall in the category of the conventional continuous optimization problems, notably, the linear and convex optimization. However, there exist cases that the objective function and/or the feasible domain are nonconvex. Then the associated decision model may have many local optimal solutions. The number of such local solutions is not known before solving the problem, and the quality of local and global solutions may vary substantially. These decision models can be very difficult for the conventional optimization methods to become directly applicable. Therefore, innovative global optimization concepts and techniques are needed.

Global optimization is a branch of applied mathematics and numerical analysis. The goal of global optimization is to find the best possible element $\mathbf{x}$ in a set $\mathcal{S}$, according to some criteria expressed in the objective function f. A good definition can be referred to

[44].

1.1 Problem Description

Consider the following global optimization problem with a single objective function:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{S}, \end{aligned} \tag{1.1}$$

where $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n | \ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u},\ g_j(\mathbf{x}) \leq 0,\ j = 1, \ldots, J\}$ is a nonempty compact subset of $\mathbb{R}^n$, with the following notations adopted:

$\mathbf{x} \in \mathbb{R}^n$: an n dimensional real vector of decision variables,

$f \in \mathbb{R}^n \to \mathbb{R}$: a real valued continuous objective function,

$\mathbf{l}$, $\mathbf{u} \in \mathbb{R}^n$: explicit lower and upper bounds on $\mathbf{x}$,

$g_j \in \mathbb{R}^n \to \mathbb{R},\ j = 1, \ldots, J$: a continuous function.

The analytical properties of f being continuous and $\mathcal{S}$ being compact guarantee, by the Weierstrass theorem of the classical analysis [101, 44], that the optimal solution set of the global optimization problem is non-empty.

When $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n | \ -\infty < \mathbf{x} < \infty\}$, the global optimization problem is usually called an unconstrained problem. When $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n | \ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$, it is called a box constrained problem. If $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n | \ A\mathbf{x} \leq \mathbf{b},\ A \in \mathbb{R}^{m \times n},\ \mathbf{b} \in \mathbb{R}^m\}$, it is a linearly constrained problem. If $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n | \ g_j(\mathbf{x}) \leq 0,\ g_j$ may be nonlinear function, $j = 1, \ldots, J\}$, it is a nonlinearly constrained problem. In this research, we focus on solving problems with explicit linear and nonlinear constraints.

In the case of optimizing a single function f, an optimum is either its maximum or minimum, depending on what we are looking for. In global optimization, it is a convention that optimization problems seek their minimizers. Throughout this research we use the minimum objective function value as the optimal value, and the definitions of local and global minima (optima) are listed below.

Global Minimum A global minimum solution of an objective function $f : \mathcal{S} \to \mathbb{R}$ is an element $\mathbf{x}^* \in \mathcal{S}$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}),\ \forall \mathbf{x} \in \mathcal{S}. \tag{1.2}$$

Neighborhood Let $\|\cdot\|$ denote the Euclidean norm in $\mathbb{R}^n$ and $\varepsilon > 0$ be a real number, then an ε-neighborhood of a given point $\mathbf{y} \in \mathbb{R}^n$ is defined as

$$N(\mathbf{y}, \varepsilon) \triangleq \{\mathbf{x} \in \mathbb{R}^n |\ \|\mathbf{x} - \mathbf{y}\| < \varepsilon\}. \tag{1.3}$$

Local Minimum A local minimum solution of an objective function $f : \mathcal{S} \to \mathbb{R}$ is an element $\mathbf{x}^{\text{loc}} \in \mathcal{S}$ with $f(\mathbf{x}^{\text{loc}}) \leq f(\mathbf{x})$ for all $\mathbf{x}$ in a neighborhood of $\mathbf{x}^{\text{loc}}$, i.e.

$$\exists\ \varepsilon > 0 \text{ such that } f(\mathbf{x}^{\text{loc}}) \leq f(\mathbf{x}),\ \forall \mathbf{x} \in \mathcal{S} \cap N(\mathbf{x}^{\text{loc}}, \varepsilon). \tag{1.4}$$

Optimal Solution Set The optimal solution set $\mathcal{S}^*$ is a subset of $\mathcal{S}$ that contains all global optimal solutions.

1.2 Importance of Global Optimization

One of the fundamental principles of our world is the search for an optimal state. Hence global optimization models can be found for numerous real world applications.

It begins in the microworld where atoms in Physics try to form bonds in order to minimize the energy of their electrons [82]. Moreover, when molecules form solid bodies during the process of freezing, they try to assume energy-optimal crystal structures. There exist many other processes driven by the laws of Physics that are of global optimization nature.

As long as life goes on, we strive for perfection in many areas. We may want to reach a maximum degree of satisfaction with the least amount of effort. In our economy, profits and sales are often maximized and costs are often minimized. Therefore, optimization is one of the oldest sciences governing our daily lives [77].

Another example is the long term financial planning problem described in [63], which is posed as a stochastic program with decision rule. The decision rule requires the purchase or sale of assets in each time stage so as to keep constant asset proportions in the portfolio composition. We seek the maximum profit while the decision rule is satisfied, such requirement leads us to a constrained nonconvex optimization problem.

The global optimization also plays an important role in the biological field which includes protein structure prediction [23]. It is the prediction of the three-dimensional structure of a protein formed by the amino acid sequences, which is one of the most significant goals pursued in Bio-informatics and Theoretical Chemistry. Scientists seek to build three-dimensional protein models that minimize suitable energy functions. Since the energy functions could be highly nonconvex, the method of global optimization is needed.

In Chemical Engineering, the study of molecular conformations is a fascinating subject and molecular mechanics is a widely used method that provides a priori accurate representations of structures and energies for molecules. The small molecules are built with the lowest potential energy. Global minimum potential energy conformations of small molecules are found by utilizing a branch and bound method in [64].

Global optimization works in scheduling problems, too. The problem is to match tasks (or people) and slots (time intervals, machines, rooms, etc.) such that every task is handled in exactly one slot with additional constraints being satisfied. If there are several feasible matchings, one that minimizes some cost or dissatisfaction measure is sought out. Simple scheduling problems such as the linear assignment problem can be formulated as linear programs and solved very efficiently, but the related quadratic assignment problem is one of the hard global optimization problems, where instances with about 30 variables are at the present limit of tractability [3].

There are many other useful applications of global optimization such as the safety verification problem, where a suboptimal local solution could falsely indicate that all safety specifications are met, leading to disastrous consequences if, in actuality, a global solution exists which provides a counterexample that violates some safety specification [57]. For many problems in Chemistry, usually only the global minimizers (of the total energy) correspond to the situations that match reality. The maximum clique problem asks for the maximal number of mutually adjacent vertices in a given graph. (For implementations in robotics, see [78].) More applications can be found in the books of Pintér [84], and Floudas and Pardalos [29].

The challenges arising in the real world applications can be modeled as a wide range of mathematical programming problems by specifying different forms of the objective function f and/or feasible region $\mathcal{S}$, including the followings:

- Linear programming: f is a linear function, and $\mathcal{S}$ is a convex set defined by a system of linear equations and inequalities.
- Quadratic programming: f is a quadratic function and $\mathcal{S}$ consists of functions that are linear or quadratic.
- D.C. programming: f can be represented as the difference of two convex functions and $\mathcal{S}$ consists of equations and inequalities that can be represented as difference of two convex functions.
- Multiplicative programming: f is the product of several convex functions and $\mathcal{S}$ consists of equations or inequalities that are represented by convex functions or functions that are product of convex functions.
- Fractional programming: f is the ratio of two real functions and $\mathcal{S}$ is convex.
- Concave minimization: f is a concave function and $\mathcal{S}$ is a convex set.
- Lipschitz optimization: f is a Lipschitz-continuous function and $\mathcal{S}$ consists of inequalities that are represented by Lipschitz continuous functions.
- Nonlinear programming: Usually f belongs to the class of twice continuously differentiable functions and $\mathcal{S}$ is a nonconvex set.
- Combinatorial optimization: Problems (partially) contain discrete decision variables in f and in $\mathcal{S}$.
- Minimax problem: f is a maximal function that has to be further minimized, while the maximum is considered over a discrete set or a convex set.

Note that the problem classes listed above are not necessarily distinct. In fact, several of them are contained by more general problem types. For detailed descriptions of most of these model types and their connections, consult the book by Pardalos and Romeijn [81] and the references therein.

1.3 Difficulties in Solving Global Optimization Problems

Global optimization problems are difficult to solve for two main reasons: the difficulty of finding solutions that satisfy the constraints, and the difficulty of locating real global optimal solutions among local optima.

For the unconstrained problems and box constrained problems, finding a feasible solution does not need too much work. However, when $\mathcal{S}$ becomes more complicated — for example, disconnected or highly nonconvex, it becomes quite challenging to obtain a feasible solution. Actually, in many problems such as linear programming problems, determining a feasible solution needs the same amount of effort as getting an optimal solution. Furthermore, the existence of constraints affects the way of searching for an optimal solution. If a method is to keep feasibility throughout the search, it has to follow certain rules restricted by the constraints. If the method violates feasibility in the searching procedure, it has to force the solutions back to the feasible region before it stops in order to produce at least a meaningful solution.

Besides the difficulties caused by the structure of the feasible region $\mathcal{S}$, distinguishing local optima and global optima is another important issue in global optimization. It deals with mathematical programming problems in the presence of multiple local optima. Observe that, typically, the number of local optima is unknown before the problem is

solved. It is highly possible that a conventional gradient-based method, e.g., Newton's method, Quasi-Newton method [32, 27], modified steepest descent method [4] or conjugate gradient method [27] may be trapped by local optima when the feasible region around the global optimum is not well-conditioned [10]. This is because all standard tools of nonlinear optimization such as the derivatives, gradients and subgradients, can at most determine local minima. They cannot tell the differences between local and global solutions.

To further illustrate some of the difficulties for a multimodal nonlinear function, we took the example from Birbil [11] and modified it to illustrate our points. Note that the shaded area in Figure 1.1 is the feasible region of the variable $\mathbf{x}$.

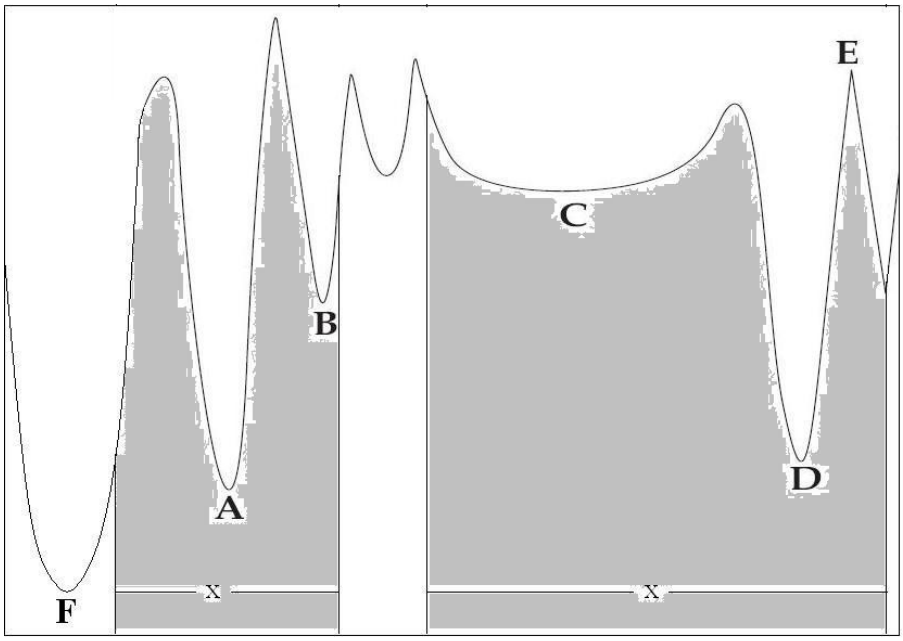

Figure 1.1: Illustration of difficulties in optimizing a multimodal nonlinear function

- The global minimizer A is in a deep valley and it is surrounded by tall hills that are difficult to overcome by the gradient-based methods.
- Point B is a promising local minimizer that is in the neighborhood of the global minimizer. A search method may be trapped in B before reaching A.
- The feasible region is disconnected, which means a solution may not be able to improve by going directly from C to B.
- A search method may spend excessive iterations before passing the shallow basin around C.
- The promising local minimizer D may mislead the search method to the regions that are far from the global minimizer A.
- The function is non-differentiable at E, hence the classical gradient-based methods can not be applied.
- The value at point F is smaller than that at A, but the constraints of $\mathbf{x}$ indicate that F is an infeasible solution so that it should be eliminated.

Due to these difficulties, the complexity of solving global optimization problems has to be addressed. For example, Murty and Kabadi showed that there exist quadratic and nonlinear programming problems that belong to the class of NP-completeness [74], and Vavasis [108] showed that, there is no efficient algorithm for solving the nonconvex programming problems in general.

1.4 Contribution of this Research

Birbil and Fang [11, 12] have developed a novel stochastic global optimization method called Electromagnetism-like method (EM) for solving the global optimization problems with bounded (box) constraints. The details of the original method will be shown in the next chapter. In this dissertation, we have further enhanced the EM method for solving linearly and convex quadratically constrained optimization problems (EM-Lin and EM-CQ). We have also tested the methods on some selected problems and implemented EM-Lin for solving a financial planning problem in the real world.

To be more specific, EM-Lin and EM-CQ have been developed with the following properties:

- The proposed methods do not require the derivative information of the objective function, they require function evaluations only.
- To start the algorithms, a number of feasible starting points are needed. EM-Lin and EM-CQ utilize some modified initialization processes to generate starting feasible points.
- We have modified the whole procedure of the original EM method while keeping its essential structure to make the new approaches more competitive in performance.
- The algorithms are designed in the way that, throughout the searching procedures, all the points visited in the search are feasible. Therefore, when the algorithms stop, they can at least provide some meaningful solutions. This also allows us to restart the algorithms directly.
- We have implemented EM-Lin for a particular long-term financial planning problem. We modeled the problem, solved it and compared the result with other known

methods. The result shows that EM-Lin is highly competitive and easy to implement.

- We have compared the proposed methods with other known global optimization methods in a relatively large set of test problems. Our results are more promising than those reported before.

- The general structures of the proposed methods are suitable for the development of parallel computation.

1.5 Outline of the Dissertation

This dissertation is organized as follows: Chapter 2 includes two parts. The first part is a literature review of different global optimization methods, with their advantages and disadvantages. Followed by is a review of the original EM method. The second part is a survey of constrained optimization, which also provides the optimality conditions for solving constrained optimization problems.

Chapter 3 introduces the EM method for linearly constrained problems (EM-Lin). Computational experiments are shown in Chapter 4. We also compare the EM method with some existing global optimization methods and point out the superiority of the EM method. In Chapter 5, we present one particular application of the EM-Lin method for long-term financial planning to show the potential of our work. The EM method for solving convex quadratically constrained problem (EM-CQ) is developed in Chapter 6. We present the numerical experiments and comparisons of EM-CQ to some existing solvers in Chapter 7. We conclude this dissertation by Chapter 8. In addition to giving a summary of the dissertation, we discuss future research areas in the last section.

Chapter 2

Literature Review

In this chapter we intend to provide a survey of various global optimization techniques and a review of constrained optimization methods. We will also give an introduction to the EM method for solving problems with bounded constraints.

2.1 Global Optimization Methods

This section provides a classification of global optimization methods.

2.1.1 Taxonomy of Global Optimization Methods

Generally, optimization methods can be divided into the deterministic and probabilistic algorithms. A deterministic algorithm does not involve random elements, while a probabilistic algorithm does. When facing a lower dimensional problem, the search space can be efficiently explored using, for example, a divide and conquer scheme. Moveover, at least in theory, a deterministic algorithm may find at least one, or all global optimal solutions.

However, if the dimensionality of the search space becomes higher and/or the objective function of the model becomes more complicated, it is harder to solve the problem in a deterministic manner. It would possibly result in an exhaustive enumeration of the search space, which makes it inapplicable. Thus, for a higher dimensional problem without any special structure, it is perhaps more practical to apply stochastic methods and intelligent heuristic approaches. In general, such methods do not have a complete convergence proof. However, in many cases, they may offer practical tools to handle problems that are out of the reach of theoretically correct, rigorous methodologies.

Figure 2.1 gives a taxonomy for global optimization methods which we will present in our review. The review consists of two parts: the first part presents the deterministic global optimization methods. The second part gives a study of stochastic methods and heuristics. In our discussion, some specific approaches are explained in details in the cited literature.

Deterministic Algorithms

This section surveys some deterministic global optimization algorithms. An algorithm is deterministic if it always yields the same results (outputs) when given the same inputs. The focus of this survey is on the general techniques that are applicable to a wide variety of combinatorial and continuous optimization problems that could arise in the real world.

Branch and Bound Method In this method (in the case of minimization), the region of all feasible solutions is repeatedly partitioned into smaller and smaller subsets according to certain rules. The subproblems are defined on these subsets, a lower bound and a upper bound are calculated within each subset by solving subproblems which are easier to handle. After each partitioning, those subsets with lower bounds that exceed

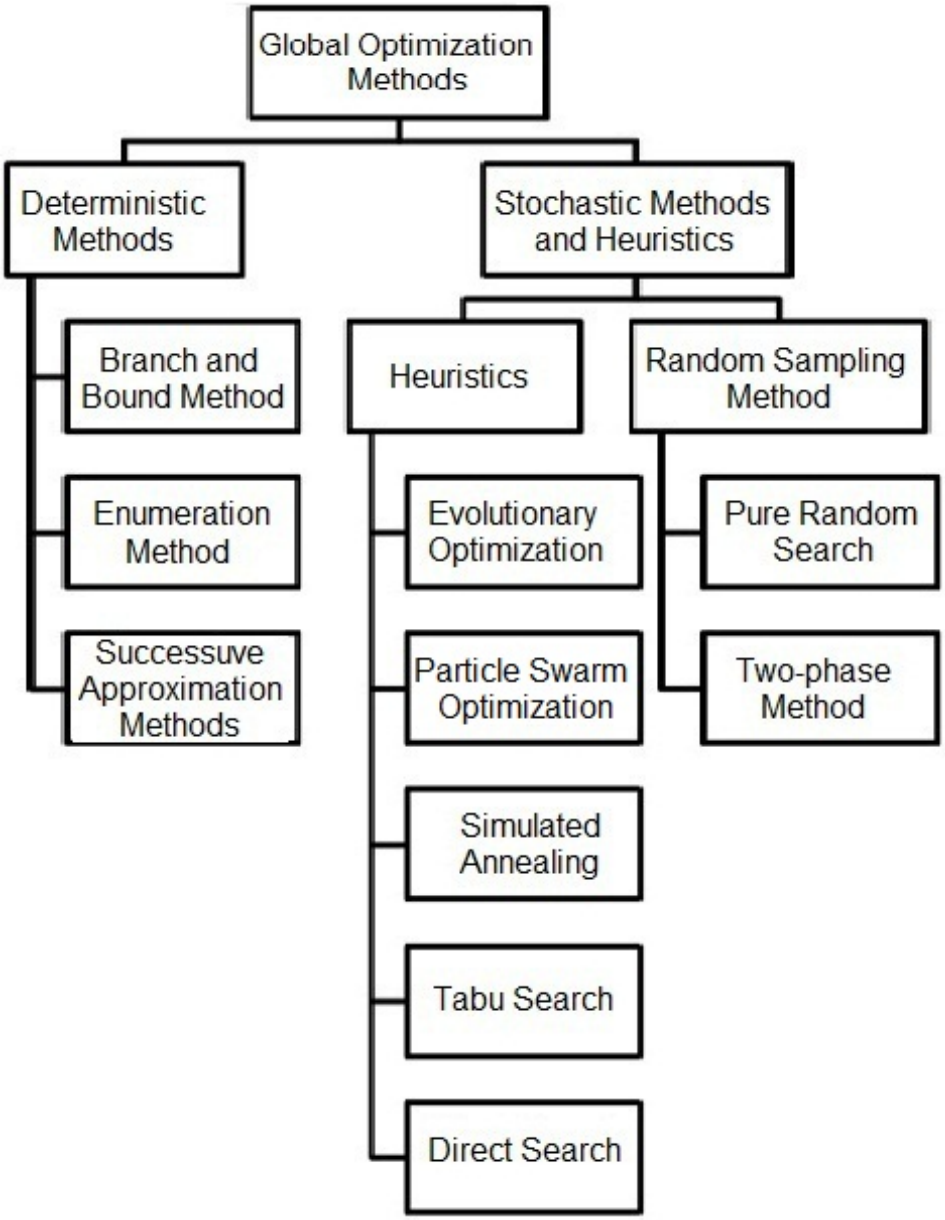

Figure 2.1: Taxonomy of global optimization methods.

the value of a known feasible solution are excluded for further consideration. The partitioning continues until a feasible solution is found such that its objective function value is no greater than the lower bound for any subset. Note that as the partitioning goes further, the number of subproblems may grow exponentially. Therefore it becomes crucial to eliminate the subsets that contain fruitless candidates to reduce the computational burden.

The method was proposed by Land and Doig in 1960 for solving mixed integer programming problems [56]. The global convergence result of this method is given in [110]. Recently it has been modified and successfully implemented for continuous global optimizations. For example, Maranas and Floudas utilized it to find the global minimum potential energy conformations in [64], Maranas et al. applied the method to solve long-term financial planning problems in [63]. For other applications of the branch and bound method, consult, e.g., [85, 76, 39, 44, 51, 84, 28, 100].

Enumeration Method This method is based upon a complete enumeration of all possible solutions. It is applicable for combinatorial optimization problems and some well-structured global optimization models involving concave programming. The disadvantage is that most of such algorithms are computationally intractable for large size problems. For more applications, refer to [44].

Successive Approximation (Relaxation) Method In this approach, the initial optimization problem is replaced by a sequence of relaxed subproblems that are easier to solve. Successive refinement of the subproblems is done by, for example, cutting planes. This approach is applicable for many global optimization problems [44].

The cutting plane method is one of the most important relaxation methods. It works

by generally cutting off the feasible regions that do not contain the optimal solutions for sure. Such procedures are popularly used to find integer solutions to mixed integer programming problems, as well as to solving general convex optimization problems. The usage of cutting planes for solving mixed integer programming problems was introduced by Ralph E. Gomory. For continuous optimization problems, it has also been shown that the cutting plane method is useful for concave minimization [9] and convex programming problems [104, 102].

Stochastic Methods

Stochastic methods usually are based on some random sampling of the feasible region. The pure random search method is introduced first. Then, the sample clustering, deterministic refinement, and statistical stopping rules can be added as enhancements to the basic scheme of pure random search. Such methods are applicable to both discrete and continuous global optimization problems under very general assumptions.

Pure Random Search Method The simplest stochastic method is the pure random search that includes the following procedures: (i) some points of the feasible region $\mathcal{S}$ are randomly sampled. The point yielding the least sampled value is kept and used as the starting point; (ii) New points of $\mathcal{S}$ are sampled according to certain rules and compared with the previous best point (the one with the least objective value); (iii) The procedure continues until some stopping criteria are satisfied. The test relies on the fact that, as the sample size increases, the probability of not improving over $\mathcal{S}$ decreases. The proof of convergence of the pure random search to the global optimum (with probability one) is shown in [6, 22, 5, 99, 83]. In general, a pure random search method is easy to implement. However, it may not be efficient, especially for problems in higher dimensions, since the

convergence speed is slow in this case.

Two-phase Method Many stochastic methods for global optimization consist of two phases: the global phase and local phase. In the global phase, just like the pure random search method, sample points are drawn from the domain $\mathcal{S}$ according to certain distributions with their objective function values calculated. In the local phase, a local optimization method is applied to the set of drawn sample points. The goal of the global phase is to obtain an approximate global extreme while the local phase is to find more precise local extrema. For more details, refer [87]. A number of known methods, such as the Newton's method and steepest descent method, can be utilized in the local phase.

The pure random search method can be viewed as a simplified two phase method. There are other methods such as the single-start method, multi-start method and clustering method [88, 94]. A modification of the pure random search that involves a local method starting from the best point is called a single-start method. A modification that consists of starting local methods from multiple sample points is called a multi-start method.

The multi-start method is a useful method, however, it may find the same local optimum many times. One class of the methods which aim to overcome this drawback and accelerate the computational speed are clustering methods [2, 40, 46, 47, 97]. Clustering global optimization methods can be viewed as a modified form of the standard multi-start procedure. The two main steps of a clustering method are: (i) sample points in the search domain and find the local minima, (ii) apply a clustering technique to identify groups representing the neighborhoods of local minima. If this procedure successfully identifies the corresponding groups, the redundant local search can be avoided by simply starting a local search from some points within each cluster.

Heuristics

Heuristic methods are experience-based techniques that are helpful for problem solving, learning and discovery, especially when the problem is complicated and solving it optimally requires too much effort. Some of these methods are inspired by careful observations of natural phenomena while some are developed merely by intuitive and practical ideas. Usually, a heuristic method produces a good solution (not necessarily optimal) or solves a simpler problem that are closely related to the given complex problem. Heuristics are used to solve some particular problems without validating its true optimality to save computational efforts.

To make heuristics work efficiently, some clever enhancements exploiting expert knowledge about the problems at hand are essential. Theoretical work on analyzing the effectiveness of useful enhancements is often lacking.

Evolutionary Algorithms Schools of Evolutionary Algorithms have evolved in the past 40 years including the genetic algorithms, developed by Holland [41], evolutionary strategies, developed by Rechenberg [86] and Schwefel [95], and evolutionary programming [30]. Each of them constitutes a different approach, however, they are inspired by the same principles of natural evolution.

Evolutionary Algorithms (EAs) are population-based meta-heuristic optimization algorithms that use biology-inspired schemes like reproduction, mutation, crossover, natural selection and survival of the fittest (measured by the objective function value) in order to refine a set of solution candidates iteratively. The ability to jump out of the regions of attractive local minimizers to the region that contains the global minimizer is enhanced by crossover and mutation. This is an advantage if, with a high probability, the crossing over and mutating rules produce offsprings with similar or even better fitness.

Therefore, the efficiency of a genetic algorithm depends on the proper selection of these rules. The tuning of a genetic algorithm requires a considerable amount of insights into the nature of the problem.

Although they are powerful for solving difficult optimization problems [93], a serious handicap of evolutionary algorithms is the heavy computational effort required by large scale problems. In addition, the structure of the encoding used in these algorithms has to be adequately designed for solving the continuous global optimization problems. For more details, please refer [68].

Particle Swarm Optimization Particle Swarm Optimization (PSO) is a population-based stochastic optimization technique developed by Eberhart and Kennedy in 1995 [26], inspired by the social behavior of bird flocking and fish schooling. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. Each particle keeps track of its coordinates corresponding to the best solution it has achieved so far. Its objective function value is also stored as $\mathbf{p}^{\text{best}}$. Another best value that is tracked by the particle swarm optimizer is the best value obtained so far by any particle in the neighborhood. This location is called $\mathbf{l}^{\text{best}}$. When a particle takes all the population as its neighbors, the best value is a global best and is denoted by $\mathbf{g}^{\text{best}}$. At each step, a particle in the population changes its velocity toward the $\mathbf{p}^{\text{best}}$ and $\mathbf{l}^{\text{best}}$ (or $\mathbf{g}^{\text{best}}$). The velocity is weighted by a random term.

In the past several years, as a relatively new paradigm, PSO has been used in a wide range of applications, for example, the training of artificial neural networks [7] and finite element updating [65].

Simulated Annealing Simulated Annealing (SA) is a technique based upon the physical analogy of cooling crystal structures that naturally arrive at a stable configuration [52]. The goal of the procedure is to minimize the objective function which represents the potential energy E. In this method, a given state with energy E_1 is compared to a state (with energy E_2) that is obtained by moving the particle to another location by a small displacement. If $E_1 \geq E_2$, the new state is accepted; while if $E_1 < E_2$, the new state is still accepted with a probability of $\exp(-(E_2 - E_1)/(kT))$, where k is the Boltzmann constant and T is the temperature of the heat bath. In this way SA can escape the entrapment at local optima. Repeating the process for a large enough number of iterations, the method is proved to be convergent with probability one [20].

The method is originally designed to solve combinatorial minimization problems in [52] and used for continuous problems in [107, 20]. Like evolutionary algorithms, the simulated annealing method is easy to understand and implement. However, in its original form, the simulated annealing method is convergent but exceedingly slow. Various enhancements of SA make it much faster. Except for simple problems, successful implementation of SA depends very much on the details of adjustments.

Tabu Search The essential idea of this heuristic is to forbid moving to points already visited in the search space, at least within the next few steps. The tabu search methodology has been primarily used for solving combinatorial optimization problems, but it can be extended to handle continuous global optimizations.

Though no rigorous convergence proof has been shown, many successful applications of tabu search are reported (e.g., in [34, 80]). However, the successful applications are given only for combinatorial optimization problems and the applications for continuous global optimization problems are still in its early stage of development.

Direct Search Direct Search methods have been known since at least the 1950s. Classical direct search methods include the Hooke and Jeeves method [43], Nelder-Mead simplex algorithm [75] and Dennis and Torczon's parallel pattern search algorithm [21]. In a direct search method, if a direction that leads us to a new solution with smaller objective function value, we enlarge the step length in hope of extending the improvement. On the other hand, if a new direction leads to a worse position (in terms of the function value), either that direction is not used, or the step length is shrunk. Based on the above concept, each method utilizes different strategy. For example, in Nelder-Mead simplex algorithm, under different conditions, there are procedures of reflection, expansion, contraction and reduction [75]. The method finds a minimal solution of a problem by repeatedly performing the above procedures.

2.1.2 General Scheme of the EM Method

Since this research proposes a solution method based on the EM method, it is important to examine carefully the original approach. The approach was proposed by Birbil and Fang [12, 13] and modified by Debels et al [19]. The original EM method works for the nonlinear optimization problems with bounded variables in the following form:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{S}, \end{aligned} \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function and

$$\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n | -\infty < l_k \leq x_k \leq u_k < \infty,\ k = 1, 2, \ldots, n\}. \tag{2.2}$$

is a bounded feasible region.

For the above problem, the following parameters are given: the dimension of the problem (n), the real valued function ($f(\mathbf{x})$) and the lower and upper bounds (l_k, u_k for $k = 1, 2, \ldots, n$). Since EM works on a set of sample points (population), there is an additional predetermined parameter, r, which denotes the number of points in the population.

The general scheme of the method is given in Algorithm 1. In this scheme an iteration of the algorithm corresponds to one pass of the **while** loop. There are four procedures: *Initialize*, *Local*, *CalcF*, and *Move*. The first procedure, *Initialize*, is used to generate a population in the feasible region and calculate their initial objective function values. *Local* is a neighborhood search procedure, which can be applied to one or many points for local refinement at each iteration. *CalcF* is a procedure to find the moving direction for each solution in the population. *Move* is a procedure to actually place each solution to a new point in the feasible region.

We use the notation, $\mathbf{x}^i \in \mathbb{R}^n$, to specify the i^{th} point (solution) of the population. Among these points at the current iteration, there is a particular one that has the best objective function value. This point is called the current best point and denoted by $\mathbf{x}^{\text{best}}$.

Initialization

The procedure *Initialize* is used to sample a certain number (r) of points randomly from the feasible domain, which is an n dimensional hyper-cube. Each coordinate of a point is assumed to be uniformly distributed between the corresponding upper bound and lower bound. After a point is sampled, the objective function value of the point is calculated and stored in $f(\mathbf{x})$. The procedure ends when all points are distributed in the space. And the point that has the best function value is stored as $\mathbf{x}^{\text{best}}$.

Algorithm 1 EM for Box-constrained Problems

r: number of sample points.
$Maxiter$: maximum number of iterations.
$Lsiter$: maximum number of local search iterations.
δ: local search parameter, $\delta \in [0, 1]$.

Initialize(r)
iteration = 1.
while iteration $< Maxiter$ **do**
  Local(δ, $Lsiter$)
  CalcF()
  Move()
  iteration = iteration + 1.
end while
Output $\mathbf{x}^{\text{best}}$ and $f(\mathbf{x}^{\text{best}})$.

Algorithm 2 Initialize(r)

for $i = 1$ to r **do**
  for $k = 1$ to n **do**
    $\lambda_k \sim U(0, 1)$.
    $x_k^i = l_l + \lambda_k(u_k - l_k)$.
  end for
  Evaluate $f(\mathbf{x}^i)$.
end for
$\mathbf{x}^{\text{best}} = \text{argmin}\{f(\mathbf{x}^i),\ i = 1, 2, \ldots, r\}$. (Break the tie with the smallest indexed minimum, if necessary.)

Local Search

The procedure *Local* is used to gather the local information at a point $\mathbf{x}^i$. There are many methods that can be used to accomplish this task. In the original method, a very simple one is applied so that we can focus on the EM algorithm itself.

The procedure iterates as follows: firstly, calculate the maximum feasible step length at $\mathbf{x}^i$. Secondly, locate some feasible neighbors of $\mathbf{x}^i$ in different directions. Calculate the objective function values of these neighbors and replace $\mathbf{x}^i$ by the one with the lowest function value as an improved solution. This procedure repeats until the iteration counter has reached a limit. The parameters, *Lsiter* and δ which are passed to this procedure, represent the number of iterations and the multiplier for the neighborhood search, respectively.

This algorithm can be applied at every point in the population or only on the current best point $\mathbf{x}^{\text{best}}$. In this research we limit the usage of *Local* only at $\mathbf{x}^{\text{best}}$.

Calculation of Total Force Vector

The computation of the total force vector is inspired by the superposition principle of electromagnetism theory [18]. In each iteration, a charge q_i of each point $\mathbf{x}^i$ is calculated according to $f(\mathbf{x}^i)$ in (2.3). The charge reflects the efficiency of the objective function value of the corresponding point in the population. The point with a higher charge has a lower function value and tends to attract other points to come closer to it, while the one with lower charge repulses other particles. The charge of particles are defined as

$$q_i = \exp\left(-n \times \frac{f(\mathbf{x}^i) - f(\mathbf{x}^{\text{best}})}{\sum_{k=1}^{r}[f(\mathbf{x}^k) - f(\mathbf{x}^{\text{best}})]}\right). \tag{2.3}$$

Then the total force vector $\mathbf{f}^i$ exerted on point $\mathbf{x}^i$ is calculated by adding together the

Algorithm 3 Local($Lsiter$, δ)

$Lsiter$: the number of maximum iterations in the local search.
δ: the multiplier of the step length.

iteration = 1.
Step length $\alpha = \delta \max_k\{u_k - l_k\},\ k = 1, 2, \ldots, n$.
for $i = 1$ to r **do**
  for $k = 1$ to n **do**
    $\lambda_1 \sim U(0, 1)$.
    while iteration $<$ $Lsiter$ **do**
      $\mathbf{y} = \mathbf{x}^i$.
      $\lambda_2 = U(0, 1)$.
      if $\lambda_1 > 0.5$ **then**
        $y_k = \min\{y_k + \lambda_2\alpha, u_k\}$.
      else
        $y_k = \max\{y_k - \lambda_2\alpha, l_k\}$
      end if
      if $f(\mathbf{y}) < f(\mathbf{x}^i)$ **then**
        $\mathbf{x}^i = \mathbf{y}$.
        iteration = $Lsiter$ - 1.
      end if
      iteration = iteration + 1.
    end while
  end for
end for
$\mathbf{x}^{\text{best}} = \text{argmin}\{f(\mathbf{x}^i), i = 1, 2, \ldots, r\}$.

individual component forces, $\mathbf{f}^{ij}$, between pairs of points $\mathbf{x}^i$ and $\mathbf{x}^j$, $j = 1, 2, \ldots, r,\ j \neq i$. The magnitude of this component force is inversely proportional to the Euclidean distance between the points and directly proportional to the product of their charges.

$$\mathbf{f}^i = \sum_{j \neq i}^{r} \mathbf{f}^{ij},\ i = 1, 2, \ldots, r, \tag{2.4}$$

where

$$\mathbf{f}^{ij} = \begin{cases} (\mathbf{x}^j - \mathbf{x}^i)\frac{q_i q_j}{\|\mathbf{x}^j - \mathbf{x}^i\|^2}, & \text{if } f(\mathbf{x}^j) < f(\mathbf{x}^i) \\ (\mathbf{x}^i - \mathbf{x}^j)\frac{q_i q_j}{\|\mathbf{x}^j - \mathbf{x}^i\|^2}, & \text{if } f(\mathbf{x}^j) \geq f(\mathbf{x}^i) \end{cases},\ i = 1, 2, \ldots, r. \tag{2.5}$$

Closely examining the algorithm, we see that the determination of a direction via the total force vector is similar to the statistical estimation of the gradient vector of f. But the Euclidean distance between two points also affects the magnitude of the force. Therefore, the points that become close enough may lead each other to a direction other than the statistically estimated gradient of f.

To avoid being trapped, a modification is performed by adding a perturbed point $\mathbf{x}^p$ which is the farthest point from the current best point $\mathbf{x}^{\text{best}}$ defined by,

$$\mathbf{x}^p = \text{argmax}\{\|\mathbf{x}^{\text{best}} - \mathbf{x}^i\|,\ i = 1, 2, \ldots, r\}. \tag{2.6}$$

At $\mathbf{x}^p$, the component forces are perturbed by a random number $\lambda \sim U(0, 1)$,

$$\mathbf{f}^{pj} = \begin{cases} (\mathbf{x}^j - \mathbf{x}^p)\frac{\lambda q_p q_j}{\|\mathbf{x}^j - \mathbf{x}^p\|^2}, & \text{if } f(\mathbf{x}^j) < f(\mathbf{x}^p), \\ (\mathbf{x}^p - \mathbf{x}^j)\frac{\lambda q_p q_j}{\|\mathbf{x}^j - \mathbf{x}^p\|^2}, & \text{if } f(\mathbf{x}^j) \geq f(\mathbf{x}^p). \end{cases} \tag{2.7}$$

Also, the directions of the component forces are perturbed. That is, if λ is less than a given parameter ν, then the direction of the component force is reversed. Consequently,

there exists one point in the population for which the direction of movement may be reversed. The purpose of introducing the perturbed point is to allow the algorithm explore more areas in the feasible region for global convergence.

Algorithm 4 CalcF(ν) ($\nu \sim U(0,1)$)

ν: the parameter for the perturbed point.

$\mathbf{x}^p = \text{argmax}\{\|\mathbf{x}^{\text{best}} - \mathbf{x}^i\|,\ i = 1, 2, \ldots, r\}$.
for $i = 1$ to r **do**
  $q_i = \exp\left(-n \times \frac{f(\mathbf{x}^i) - f(\mathbf{x}^{\text{best}})}{\sum_{k=1}^{r}[f(\mathbf{x}^k) - f(\mathbf{x}^{\text{best}})]}\right)$.
  $\mathbf{f}^i = \mathbf{0}$.
end for
for $i = 1$ to r **do**
  for $j = 1$ to r **do**
    if $i \neq j$ **then**
      if $i \neq p$ **then**
        $\mathbf{f}^{ij} = (\mathbf{x}^j - \mathbf{x}^i)\frac{q_i q_j}{\|\mathbf{x}^j - \mathbf{x}^i\|^2}$.
      else
        $\lambda \sim U(0,1)$.
        $\mathbf{f}^{ij} = (\mathbf{x}^j - \mathbf{x}^i)\frac{\lambda q_i q_j}{\|\mathbf{x}^j - \mathbf{x}^i\|^2}$.
        if $\lambda < \nu$ **then**
          $\mathbf{f}^{ij} = -\mathbf{f}^{ij}$ (reverse direction).
        end if
      end if
      if $f(\mathbf{x}^j) < f(\mathbf{x}^i)$ **then**
        $\mathbf{f}^i = \mathbf{f}^i + \mathbf{f}^{ij}$ (attraction).
      else
        $\mathbf{f}^i = \mathbf{f}^i - \mathbf{f}^{ij}$ (repulsion).
      end if
    end if
  end for
end for
Output the total force vectors $\{\mathbf{f}^1, \mathbf{f}^2, \ldots, \mathbf{f}^r\}$.

Movement According to Total Force Vector

After evaluating the total force vector $\mathbf{f}^i$, the point $\mathbf{x}^i$ moves along the direction of the force by a random step length λ that is uniformly distributed between 0 and 1, i.e.,

$$\mathbf{x}^i_{\text{new}} = \mathbf{x}^i + \lambda \frac{\mathbf{f}^i}{\|\mathbf{f}^i\|} \circ \mathbf{RNG},\ i = 1, 2, \ldots, r,\ i \neq best, \tag{2.8}$$

where $\mathbf{x}^i_{\text{new}}$ is the updated point and $\mathbf{RNG} \in \mathbb{R}^n$. In (2.8), $\frac{\mathbf{f}^i}{\|\mathbf{f}^i\|} \circ \mathbf{RNG}$ is the one-to-one matching multiplication between $\frac{\mathbf{f}^i}{\|\mathbf{f}^i\|}$ and $\mathbf{RNG}$. If f^i_j, the j^{th} component of $\mathbf{f}^i$, is nonnegative, RNG_j represents the distance between the j^{th} component of $\mathbf{x}^i$ and the j^{th} component of the upper bound. If $f^i_j < 0$, RNG_j represents the distance between the j^{th} component of $\mathbf{x}^i$ and the j^{th} component of the lower bound. The details are shown in the algorithm 5. Note that the best point $\mathbf{x}^{\text{best}}$ is not moved.

Algorithm 5 Move($\mathbf{f}$)

$\mathcal{F} = \{\mathbf{f}^1, \mathbf{f}^2, \ldots, \mathbf{f}^r\}$: the set of total force vectors exerted on $\mathbf{x}^i,\ i = 1, 2, \ldots, r$.

for $i = 1$ to r **do**
  if $i \neq best$ **then**
    $\lambda \sim U(0, 1)$.
    $\mathbf{f}^i = \frac{\mathbf{f}^i}{\|\mathbf{f}^i\|}$.
    for $k = 1$ to n **do**
      if $f^i_k > 0$ **then**
        $x^i_k = x^i_k + \lambda f^i_k (u_k - x^i_k)$.
      else
        $x^i_k = x^i_k + \lambda f^i_k (x^i_k - l_k)$.
      end if
    end for
  end if
end for
Output the updated points $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^r\}$.

Termination Criteria

In the original EM method the procedure is terminated by using a predetermined maximum number of iterations. According to the test results, in general, 25 iterations per dimension (i.e., $Maxiter = 25n$ where n is the dimension of the problem) is satisfactory for converging to the optimum point for those moderate difficulty functions.

Another termination criterion that might be used is the successive number of iterations spent without changing the current best point. In other words, if the current best point remains for a certain number of iterations, the algorithm may be stopped. In our new algorithm (EM-Lin), this criterion will be applied. Several other stopping conditions are studied in [103]. One of the frequently used criteria for testing algorithms is to terminate the algorithm when the observed objective function value is ε-close to the optimal value [45]. However, this criterion is not appropriate if the global optimum is not known in advance.

2.2 Constrained Optimization Methods

Constrained optimization has an objective function with some explicit constraints to be satisfied. Constraints can be either in the form of equations or inequalities.

A general nonlinear programming problem is in the following form:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & h_i(\mathbf{x}) = 0,\ i = 1, 2, \ldots, m \\ & g_j(\mathbf{x}) \leq 0,\ j = 1, 2, \ldots, p \\ & \mathbf{x} \in \mathbb{R}^n, \end{aligned} \tag{2.9}$$

where $m \leq n$ and the function $f,\ h_i$ and g_i are of real values. We introduce the vector-valued function $\mathbf{h} = (h_1, h_2, \ldots, h_m)$ and $\mathbf{g} = (g_1, g_2, \ldots, g_p)$ and rewrite the problem as

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{x} \in \mathbb{R}^n. \end{aligned} \tag{2.10}$$

Strict convexity of the objective function f is not sufficient to guarantee a unique minimum. In addition, the feasible domain should be convex to guarantee that the problem has a unique solution.

Because of the constraints, the stationary points of $f(\mathbf{x})$ alone may not be feasible. In fact, the optimal solutions to the constrained problems are often not stationary points of the objective functions. Consequently, the technique of searching for all stationary points of the objective function that also satisfy the constraints does not work. Thus, the necessary conditions for local minimizers are developed in order to reduce the set of possible solutions, and some methods based on the necessary conditions are proposed.

The rest of this section is organized as follows. First, the necessary conditions that should be satisfied by an extreme point are listed. Then, on the basis of necessary conditions, we introduce several methods including the primal methods, penalty and barrier function methods and dual methods for solving the constrained optimization problems.

2.2.1 Constrained Minimization Conditions

Consider the problem with equality constraints

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{x} \in \mathbb{R}^n. \end{aligned} \tag{2.11}$$

The *Lagrangian* of a constrained optimization problem is defined to be

$$l(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}), \tag{2.12}$$

where $\mathbf{x} \in \mathbb{R}^n$ and $\boldsymbol{\lambda} \in \mathbb{R}^m$.

We now define the regular points of equality constraints.

Regular point for equality constraints Let $\mathbf{x}^*$ be a point satisfying the constraints

$$\mathbf{h}(\mathbf{x}^*) = \mathbf{0}. \tag{2.13}$$

Then $\mathbf{x}^*$ is said to be a regular point of the constraints (2.13) if the gradient vectors $\nabla h_i(\mathbf{x}^*), i = 1, \ldots, m$, are linearly independent.

The following theorems give the first order and second order necessary conditions for a point to be a local minimal point subject to equality constraints.

Lemma 1 (First and second order necessary conditions) [62] Let $\mathbf{x}^*$ be a local optimal solution of f subject to the constraints $\mathbf{h}(\mathbf{x}^*) = \mathbf{0}$. Assume that $\mathbf{x}^*$ is a regular

point. Then there is a $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that,

$$\nabla f(\mathbf{x}^*) + \boldsymbol{\lambda}^T \nabla \mathbf{h}(\mathbf{x}^*) = \mathbf{0}, \tag{2.14}$$

or equivalently,

$$\nabla_{\mathbf{x}} l(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{0}, \tag{2.15}$$

$$\nabla_{\boldsymbol{\lambda}} l(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{0}. \tag{2.16}$$

Furthermore, the matrix

$$L(\mathbf{x}^*) = F(\mathbf{x}^*) + \boldsymbol{\lambda}^T H(\mathbf{x}^*) \tag{2.17}$$

is positive semi-definite on the tangent plane

$$\mathcal{M}_1 = \{\ \mathbf{y} \in \mathbb{R}^n \mid \nabla \mathbf{h}(\mathbf{x}^*)\mathbf{y} = 0\}, \tag{2.18}$$

where F and H are the Hessians of f and $\mathbf{h}$, respectively.

Now consider the problem with inequality constraints

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{x} \in \mathbb{R}^n, \end{aligned} \tag{2.19}$$

where f, $\mathbf{h}$ and $\mathbf{g} \in C^k$ with $k \geq 1$. The necessary conditions for a local minimal point are given below.

Here we define a regular point for inequality constraints.

Regular point for inequality constraints Let $\mathbf{x}^*$ be a point satisfying the constraints

$$\mathbf{h}(\mathbf{x}^*) = \mathbf{0},\ \mathbf{g}(\mathbf{x}^*) \leq \mathbf{0}. \tag{2.20}$$

Let $\mathcal{J}$ be the set of indices j for which $g_j(\mathbf{x}^*) = 0,\ j = 1, \ldots, p$. Then $\mathbf{x}^*$ is said to be a regular point of the constraints (2.20) if the gradient vectors $\nabla h_i(\mathbf{x}^*),\ 1 \leq i \leq m$ and $\nabla g_j(\mathbf{x}^*),\ j \in \mathcal{J}$ are linearly independent.

Lemma 2 (Karush-Kuhn-Tucker (KKT) conditions) [62] Let $\mathbf{x}^*$ be a relative minimum point for problem (2.19). Suppose $\mathbf{x}^*$ is a regular point. Then there exist vectors $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^p$ with $\boldsymbol{\mu} \geq \mathbf{0}$, such that

$$\nabla f(\mathbf{x}^*) + \boldsymbol{\lambda}^T \nabla \mathbf{h}(\mathbf{x}^*) + \boldsymbol{\mu}^T \nabla \mathbf{g}(\mathbf{x}^*) = \mathbf{0}, \tag{2.21}$$

$$\boldsymbol{\mu}^T \nabla \mathbf{g}(\mathbf{x}^*) = \mathbf{0}. \tag{2.22}$$

Lemma 3 (Second order necessary conditions) [62] Suppose that f, $\mathbf{h}$, $\mathbf{g} \in C^2$ and $\mathbf{x}^*$ is a regular point. If $\mathbf{x}^*$ is a relative minimum point of the problem (2.19), then there exist $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^p$ with $\boldsymbol{\mu} \geq \mathbf{0}$, such that (2.21) and (2.22) hold and the Hessian matrix

$$L(\mathbf{x}^*) = F(\mathbf{x}^*) + \boldsymbol{\lambda}^T H(\mathbf{x}^*) + \boldsymbol{\mu}^T G(\mathbf{x}^*), \tag{2.23}$$

is positive semi-definite on the subspace

$$\mathcal{M}_2 = \{\ \mathbf{y} \in \mathbb{R}^n \mid \nabla \mathbf{h}(\mathbf{x}^*)\mathbf{y} = 0,\ \nabla g_j(\mathbf{x}^*)\mathbf{y} = 0 \text{ for all } j \in \mathcal{J}\}, \tag{2.24}$$

where F, H and G are the Hessians of f, $\mathbf{h}$ and $\mathbf{g}$, respectively, and

$$\mathcal{J} = \{\ j \mid g_j(\mathbf{x}^*) = 0,\ \mu_j > 0,\ j = 1, \ldots, p\}. \tag{2.25}$$

After introducing the necessary conditions for local minimum solutions, we are going to bring in some practical methods for solving the constrained optimization problems. Some of these methods are designed based on the necessary conditions.

2.2.2 Primal Methods

A primal method directly works on the original problem by searching through the feasible region for an optimal solution. Each candidate in the process is feasible and the value of the objective function usually decreases iteration by iteration. There are several advantages of the primal methods. First, if the process terminates before reaching the solution, the terminating point is still feasible. The other advantage is that most primal methods are more applicable for general nonlinear programming problems. However, the primal methods can lead to some difficulties caused by the feasible domain, and they may lack the global convergence proof.

Some commonly seen primal methods include the feasible direction method, active set method and gradient projection method.

Feasible Direction Method

The idea of feasible direction method is to take steps through the feasible region of the form

$$\mathbf{x}^{i+1} = \mathbf{x}^i + \alpha_i \mathbf{d}^i, \tag{2.26}$$

where $\mathbf{d}^i \in \mathbb{R}^n$ is a direction vector and α_i is a nonnegative scalar used as the step length. A feasible direction method can be considered as a natural extension of the unconstrained descent method. Each step is the composition of selecting a feasible direction $\mathbf{d}^i$ and a constrained line search for an appropriate step length $\alpha_i > 0$. The algorithm is originally proposed by Zoutendijk [115] and has been modified over the last few decades. For more information, please refer [62, 54].

Active Set Method

Problems with inequality constraints can be approached with an active set method. The idea of the active set method is to partition the inequality constraints into two groups: those are active and those are inactive. The inactive constraints are ignored since the points can move in its neighborhood without violating the inactive constraints. A working set is defined as the index set of active constraints. By dynamically adding and dropping constraints from the working set, the correct set of active constraints is determined in the search process. Convergence cannot be guaranteed for many of this approach because the working set may change all the time. But in practice the active set method with various refinements are often very effective. Details the of active set method are described in [62, 79, 73].

Gradient Projection Method

The idea of the gradient projection method comes from the method of steepest descent for unconstrained problems. In this method, the negative gradient of a current point is projected onto the working surface (which is defined by the active constraints) in order to define the direction of movement. The main difficulty of this method is to keep the solution staying in the feasible region. The method has some successful implementation-

s, and found to be effective in solving general nonlinear programming problems. The method is proposed by Rosen in [90] to solve nonlinear programming problems with linear constraints. Later it is generalized to solve nonlinear programming problems with nonlinear constraints in [91].

2.2.3 Dual Methods

Dual methods are based on the fact that the solution point of a constrained problem is determined by the Lagrangian multipliers which are the unknowns associated with the problem. Therefore, instead of solving the primal problem, a dual method tries to find the Lagrangian multipliers for the dual problem.

Lemma 4 (Local duality theorem for equality constraints) [62] Suppose that the problem

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{x} \in \mathbb{R}^n,\ \mathbf{h}(\mathbf{x}) \in \mathbb{R}^m, \end{aligned} \tag{2.27}$$

has a local minimum $\mathbf{x}^*$ with the corresponding Lagrangian vector $\boldsymbol{\lambda}^* \in \mathbb{R}^m$. Suppose that $\mathbf{x}^*$ is a regular point and the corresponding Hessian matrix $L(\mathbf{x}^*)$ in (2.17) is positive definite. Define the dual function

$$\phi(\boldsymbol{\lambda}) = \min_{\mathbf{x}} \{f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x})\},\ \boldsymbol{\lambda} \in \mathbb{R}^m, \tag{2.28}$$

where the minimum is taken locally in the neighborhood of $\mathbf{x}^*$. Then the dual problem

$$\max \ \phi(\boldsymbol{\lambda}) \tag{2.29}$$

has a local solution at $\boldsymbol{\lambda}^*$ with $\mathbf{x}^*$ being the solution of (2.28) for $\phi(\boldsymbol{\lambda}^*)$.

Lemma 5 (Local duality theorem for inequality constraints) [62] Suppose that the problem

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{x} \in \mathbb{R}^n, \ \mathbf{h}(\mathbf{x}) \in \mathbb{R}^m, \ \mathbf{g}(\mathbf{x}) \in \mathbb{R}^p, \end{aligned} \tag{2.30}$$

has a local minimum $\mathbf{x}^*$ with the corresponding Lagrangian vectors $\boldsymbol{\lambda}^* \in \mathbb{R}^m$ and $\boldsymbol{\mu}^* \in \mathbb{R}^p, \ \boldsymbol{\mu}^* \geq \mathbf{0}$. Suppose that $\mathbf{x}^*$ is a regular point and the corresponding Hessian matrix $L(\mathbf{x}^*)$ in (2.23) is positive definite. Define the dual function

$$\phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \min_{\mathbf{x}} \{ f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}) \}, \ \boldsymbol{\lambda} \in \mathbb{R}^m, \ \boldsymbol{\mu} \in \mathbb{R}^p, \tag{2.31}$$

where the minimum is taken locally in the neighborhood of $\mathbf{x}^*$. Then the dual problem

$$\max \ \phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) \tag{2.32}$$

achieves a local solution at $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^* \geq \mathbf{0}$ with $\mathbf{x}^*$ being the solution of (2.31) for $\phi(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$.

From the theorems we can see that $\mathbf{x}$ is treated as the function of Lagrangian mul-

tipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$. Therefore, when we find the local maximizer of the dual problem, the solution to the original problem $\mathbf{x}^*$ can be obtained. Details of the dual methods are presented in [62]. In [36, 58, 105], the dual methods have been proposed to solve quadratic programming problems.

2.2.4 Penalty and Barrier Function Methods

Penalty and barrier function methods are similar methods that utilize different forms of cost functions which are added to the original objective functions.

Penalty Method Penalty method approximates a constrained problem by an unconstrained one by assigning a high cost to points that are not feasible. As the approximation is made more exact (by adjusting the penalty parameter), the solution of the unconstrained penalty problem approaches the solution of the original problem. In other words, a penalty method may use a sequence of exterior methods and the feasibility is obtained only at the optimum.

The advantage of the penalty method is that it is less likely to be stuck in a feasible pocket with a local minimum. Furthermore, the penalty method is more robust because in practice one may often have an infeasible starting point. However, penalty functions typically require more function evaluations. Details of the penalty method are described in [62, 98].

Barrier Method Barrier method, which is also called the interior-point method, on the other hand, approximates a constrained problem by an unconstrained problem by assigning a high cost to the points near the boundary of the feasible region. As the approximation is made more exact, the solution of the unconstrained barrier problem

approaches the solution of the original constrained problem from the interior of the feasible region. So the barrier method is characterized by its ability of preserving feasibility at all times. This is the advantage of the barrier method because even if it does not converge, one will still have a feasible solution. Details of the barrier method can be found in [14, 62].

The interior-point approach has been inspired by Karmarkar's algorithm, developed by Narendra Karmarkar in 1984 for linear programming [50]. Among all the interior-point methods, the class of primal-dual path-following interior-point methods is considered the most successful. Current efficient implementations are mostly based on a predictor-corrector technique [66]. [111] gives a survey of the interior-point methods.

Chapter 3

Electromagnetism-like Method for Linearly Constrained Problems (EM-Lin)

A novel algorithm called Electromagnetism-like Mechanism (EM) was introduced in [13]. Its general scheme has been introduced in the previous chapter for solving optimization problems with bounded variables. The objective of this research is to adapt the concept of the EM method to develop an algorithm for solving global optimization problems with more general constraints.

In this chapter, our goal is to extend the EM method for solving linearly constrained problems. The new method is called EM-Lin. The problems with linear constraints are considered because the linear constraints may be the simplest beyond the bounded constraints. Moreover, further extensions may build on this development. The details of the proposed EM-Lin will be presented in the following sections.

3.1 Outline of EM-Lin

We address the linearly constrained optimization problem in the following form:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \overline{A}\mathbf{x} \leq \overline{\mathbf{b}} \\ & \mathbf{x} \in \Omega, \end{aligned} \tag{3.1}$$

where $\Omega = \{\mathbf{x} \in \mathbb{R}^n |\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\},\ \mathbf{l}, \mathbf{u} \in \mathbb{R}^n$, $\overline{A} \in \mathbb{R}^{m_0 \times n}$ and $\overline{\mathbf{b}} \in \mathbb{R}^{m_0}$. In this case, the feasible region $\mathcal{S}$ becomes

$$\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n |\ \overline{A}\mathbf{x} \leq \overline{\mathbf{b}},\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}. \tag{3.2}$$

To simplify the formulation, we may combine the box constraints and linear constraints together. Let

$$A = \begin{bmatrix} \overline{A} \\ \vdots \\ -\mathbf{e}^i \\ \vdots \\ \mathbf{e}^i \\ \vdots \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} \overline{\mathbf{b}} \\ \vdots \\ -l_i \\ \vdots \\ u_i \\ \vdots \end{bmatrix},\ i = 1, 2, \ldots, n. \tag{3.3}$$

where $\mathbf{e}^i$ is a row vector with 1 being its i^{th} element and 0 otherwise. Then,

$$\begin{aligned} \mathcal{S} \ = \ & \{\mathbf{x} \in \mathbb{R}^n |\ \overline{A}\mathbf{x} \leq \overline{\mathbf{b}}\} \cap \{\mathbf{x} \in \mathbb{R}^n |\ -\mathbf{e}^i\mathbf{x} \leq -l_i,\ i = 1, 2, \ldots, n\} \\ & \cap \{\mathbf{x} \in \mathbb{R}^n |\ \mathbf{e}^i\mathbf{x} \leq u_i,\ i = 1, 2, \ldots, n\}. \end{aligned} \tag{3.4}$$

Therefore, by comparing the formulas of (3.3) and (3.4), we have

$$\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b}\}. \tag{3.5}$$

The equivalent form of (3.1) then becomes

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b}. \end{aligned} \tag{3.6}$$

We assume that $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$.

Our goal is to design an algorithm seeking for the global solutions while maintaining the feasibility of the solution in each iteration. In this way, the algorithm always provides a meaningful solution even when it stops prematurely.

EM-Lin contains four major steps: *Initialize*, *Local*, *CalcF* and *Move*. Comparing with the original EM method, the first step, *Initialize*, needs to be replaced by a new procedure, since the known one is no longer applicable for the linearly constrained case. A direct search is implemented in the *Local* step to achieve a better performance. The *CalcF* procedure works well for the new case and we retain it. The original *Move* procedure may produce points which violate the linear constraints. Therefore, some adjustments are needed. New terminating criteria are also added to make the algorithm more efficient.

We describe below in Algorithm 6 the main structure of the revised EM algorithm for solving linearly constrained problems of (3.6).

In the next three sections we will focus on the following aspects: the technique of *Generating Initial Feasible Population* (line 5, Algorithm 6), the *Local Search* Method (line 16, Algorithm 6) and the *Move* procedure (line 22, Algorithm 6). The procedures *Calculating Total Force* and *Determining Stopping Criteria* are also included.

Algorithm 6 EM for linearly constrained Problem

Define Parameters. Set up the initial parameters.
Δ_{tol}, Δ: the stopping tolerance and the initial stopping parameter, respectively.
$\phi > 1$, $0 < \theta < 1$: the increasing and the decreasing factor, respectively.
r: the initial population size.
Initialize. Randomly generate the initial feasible population $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^r\}$.
Evaluate the initial active point $f(\mathbf{x}^i),\ i = 1, 2, \ldots, r$.
$\mathbf{x}^{\text{best}} = \mathbf{x}^1,\ f^{\text{best}} = f(\mathbf{x}^{\text{best}})$.
Set $Successful\ improvement = true$.
while the stopping criteria are not satisfied **do**
  Set $\mathbf{x}^{\text{best}} = \text{argmin}\{f(\mathbf{x}^i),\ i = 1, 2, \ldots, r\}$.
  if $f(\mathbf{x}^{\text{best}}) < f^{\text{best}}$ **then**
    Set $f^{\text{best}} = f(\mathbf{x}^{\text{best}})$, set $Successful\ improvement = true$.
    $\Delta = \phi \times \Delta$ (increase the stopping parameter).
    Skip the **Local Search** step and go to **CalcF(ν)**.
  else
    Apply **Local Search** at $\mathbf{x}^{\text{best}}$, which provides the updated $\mathbf{x}^{\text{best}}$ and updated $Successful\ improvement$.
    if $Successful\ improvement = false$ **then**
      $\Delta = \theta \times \Delta$ (decrease the stopping parameter).
    end if
  end if
  CalcF(ν).
  Move.
  Check Stopping Criterion. (Check Section 3.6 for details.)
end while
Return $\mathbf{x}^{\text{best}}$ and f^{best}.

3.2 Initialization

Before starting the main loop of the algorithm, a set of initial points that lie inside the polyhedron

$$\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n | \ A\mathbf{x} \leq \mathbf{b}\} \tag{3.7}$$

need to be generated.

When only simple bounds (that is, $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$ with $\mathbf{l}$ and $\mathbf{u}$ being the lower and upper bounds) are present in the problem, an initial feasible population can be randomly generated by using a uniform distribution. However, generating an initial feasible population for a linearly constrained problem is not that simple. There are several ways to accomplish the task, and our algorithm considers all of them.

For generating a feasible population satisfying the linear constraints, one strategy would be firstly to ignore the linear constraints and randomly generate points. Then a newly generated point is accepted if it satisfies the constraints. This strategy is straightforward and easy to implement. However, such a strategy may not be efficient for generating a diverse feasible population. When the polyhedron formed by the linear constraints is relatively small compared to the region formed by the simple bounds, the probability that a randomly generated point falls in the polyhedron would be low. We keep this method of generating the initial feasible population in our algorithm and use it when other methods fail to provide the initial feasible population.

The second method for generating initial feasible population works as described below. Since the constraint functions are all linear, we may produce a linear programming problem by using the feasible domain $\mathcal{S}$ and an artificial linear function as the objective function. Then we apply the simplex method or interior point method to solve the problem. During the solving procedure, by recording the location of the point in each it-

eration, we are able to obtain some feasible solutions to the problem. Finally, the convex combinations of these solutions can be used as the initial feasible points. Since the linear programming problem can be solved in polynomial time by the interior point method, this method may finish generating the initial feasible population in polynomial time.

The third way of providing an initial feasible population is explained below. First, find an interior point $\mathbf{x}^0$ that lies inside the feasible domain $\mathcal{S}$. This can be done in several ways, one of which is solving a linear programming problem,

$$\begin{aligned} \max \quad & t \\ \text{s.t.} \quad & A\mathbf{x} + t\mathbf{e} \leq \mathbf{b} \end{aligned} \tag{3.8}$$

with the optimal solution $(\mathbf{x}^*, t^*)$, where $\mathbf{e} = (1, 1, \ldots, 1)^T \in \mathbb{R}^m$. If $t^* > 0$, $\mathbf{x}^*$ can be accepted as $\mathbf{x}^0$. Then, from $\mathbf{x}^0$, a set of random vectors $\{\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^q\}$ pointing to different directions are generated. Before extending these vectors to hit the boundary of $\mathcal{S}$, if we choose the step length α_i carefully, the points

$$\mathbf{x}^i = \mathbf{x}^0 + \alpha_i \mathbf{v}^i, \ \alpha_i > 0, \ i = 1, 2, \ldots, q, \tag{3.9}$$

and their convex combinations can be treated as the initial feasible points. This method has been used in some algorithms, for example, the Genetic Algorithm solver and Pattern Search solver in the MATLAB toolbox. Some modifications have been done in these solvers and the directions are more diverse than the ones which are randomly generated.

Another strategy for generating the initial feasible population is also adopted in our algorithm. The idea is used in [109]. In the first step of this strategy, we calculate the maximum-volume ellipsoid which is inscribed in the feasible region $\mathcal{S}$. Then, an initial population within the ellipsoid is randomly generated. Since the ellipsoid is inside the

feasible region, the points generated are feasible. Furthermore, the maximum-volume ellipsoid helps us to distribute the points as diverse as possible.

The computation of the maximum-volume ellipsoid inscribed inside the feasible region is carried out by the interior point method developed by Zhang and Gao [113].

In their paper, a polytope $\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n \mid C\mathbf{x} \leq \mathbf{d}\}$ is considered as the feasible region, where $C \in \mathbb{R}^{p \times n}$, $p > n$, and $\mathbf{d} \in \mathbb{R}^p$. Two assumptions are made throughout the paper,

1. The matrix C has full column rank and contains no zero rows.

2. There exists a strictly interior point $\bar{\mathbf{x}} \in \mathcal{P}$ such that $C\bar{\mathbf{x}} < \mathbf{d}$.

To produce C and $\mathbf{d}$ which satisfy the assumptions, some preprocesses are needed. Consider the assumption 1 first, if $A \in \mathbb{R}^{m \times n}$ in the original feasible region $\mathcal{S}$ meets the requirement that $m > n$ and has full column rank, the first assumption is satisfied. A and $\mathbf{b}$ in (3.6) can be used as C and $\mathbf{d}$. If $m \leq n$, artificial bounds are added to make the new constraint matrix C have more rows than columns and have the full column rank. Then the calculation of the inscribed ellipsoid is doable. Since the new feasible region is contained in $\mathcal{S}$, the generated points are feasible.

To satisfy the second assumption, we solve a linear programming problem,

$$\begin{aligned} \max \quad & t \\ \text{s.t.} \quad & C\mathbf{x} + t\mathbf{e} \leq \mathbf{d}, \end{aligned} \tag{3.10}$$

where $\mathbf{e} \in \mathbb{R}^p$ is a column vector with 1 at all positions. Suppose that the solution of (3.10) is $(\mathbf{x}^*, t^*)$. If $t^* > 0$, the assumption can be satisfied and $\mathbf{x}^*$ is used as a starting point of the calculation in following section. If $t^* = 0$, we turn to other methods of generating the initial feasible population.

To find a maximum-volume ellipsoid inscribed in $\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n \mid C\mathbf{x} \leq \mathbf{d}\}$, one needs a center $\mathbf{x}^c$ and a nonsingular scaling matrix $E \in \mathbb{R}^{n \times n}$. Then an ellipsoid can be defined as

$$\xi(\mathbf{x}^c, E) = \{\mathbf{v} \in \mathbb{R}^n \mid \mathbf{v} = \mathbf{x}^c + E\mathbf{s},\ \|\mathbf{s}\| \leq 1\}, \tag{3.11}$$

where $\|.\|$ is the Euclidean norm in $\mathbb{R}^n$.

The volume of the ellipsoid $\xi(\mathbf{x}^c, E)$ is defined to be

$$\text{Vol}(\xi) = V_n \det(E), \tag{3.12}$$

where V_n is the volume of the n-dimensional unit ball. Then, a convex programming problem *MAX-VOL* is constructed as in [113]:

$$\begin{aligned} \min_{\mathbf{x},E} \quad & -\det(E) \\ \text{s.t.} \quad & \mathbf{c}^j\mathbf{x} + \|E\mathbf{c}^j\| \leq d_j,\ j = 1, 2, \ldots, p \\ & E \in \mathcal{S}_+^n, \end{aligned} \tag{3.13}$$

where $\mathbf{c}^j$ is the j^{th} row of C, d_j is the j^{th} element of $\mathbf{d}$ and $\mathcal{S}_+^n$ represents the cone of all symmetric positive definite matrices in $\mathbb{R}^n$. Problem (3.13) is a convex programming problem and it has a unique solution $(\mathbf{x}^{c*}, E^*) \in \mathbb{R}^n \times \mathcal{S}_+^n$.

In [113], an interior point method is used to solve the problem. According to the paper, the interior point method is well-defined, the solution is unique and the algorithm is globally convergent. Though it is unlikely that polynomial convergence could be proven for the algorithm, it is fast enough in experiments. Furthermore, the method described in the above paper is easy to incorporate into our algorithm.

Once the center $\mathbf{x}^{c*}$, the scaling matrix E^* and the maximum-volume ellipsoid $\xi(\mathbf{x}^{c*}, E^*)$

are obtained in the above step, the initial feasible points can be generated within $\xi(\mathbf{x}^{c*}, E^*)$ using the following equation:

$$\mathbf{x}^i = \mathbf{x}^{c*} + E^* \boldsymbol{\eta}^i, \ i = 1, 2, \ldots, r, \tag{3.14}$$

where $\boldsymbol{\eta}^i \in \mathbb{R}^n$ and $\|\boldsymbol{\eta}^i\| \leq 1, \ i = 1, 2, \ldots, r$, are vectors generated along different coordinates.

In our algorithm, the method of finding the maximum volume ellipsoid is first applied since it may generate diverse initial solutions. If the interior point method is unable to solve the *MAX-VOL* problem (3.13) in a certain number of iterations, we turn to use the third method which is described previously. When the calculated step lengths in the third method are too small, which means the generated points are too close to $\mathbf{x}^0$ in (3.9), the second method is applied. Finally, if the second method still does not provide enough initial feasible solutions, the first method has to be utilized, though it appears to be inefficient.

3.3 Local Search

The procedure *Local* is used to gather the local information of a point $\mathbf{x}^i$ and find better solutions in the neighborhood of $\mathbf{x}^i$. As stated in [12, 13], many powerful local search methods can be utilized for the *Local* step. In this dissertation a direct search method is applied at the current best point $\mathbf{x}^{\text{best}}$. In each iteration of the direct search method, we evaluate the objective function value of each selected point in the neighborhood of $\mathbf{x}^{\text{best}}$. The new points in the neighborhood are obtained by adding to $\mathbf{x}^{\text{best}}$ a set of feasible directions $\{\mathbf{j}^i, \ i = 1, 2, \ldots, k\}$. Then we keep the one with the lowest function value as

Algorithm 7 Initialize($\mathcal{S}, r$)

$\mathcal{S}$: the feasible region given by the problem.
r: size of the initial population.

1: Set N_2, N_3 and N_4 as the thresholds for methods $2, 3$ and 4, respectively.
2: Apply method 4.
3: **if** fail to solve *MAX-VOL* in N_4 iterations **then**
4: Apply method 3. Stop when the # of initial feasible solutions $= r$.
5: **if** in method 3 the step length is smaller than N_3 **then**
6: Apply method 2. Stop when the # of initial feasible solutions $= r$.
7: **if** the # of recorded feasible solutions is smaller than N_2 **then**
8: Apply method 1. Stop when the # of initial feasible solutions $= r$.
9: **if** the # of initial feasible solutions $< r$ **then**
10: Return error.
11: **end if**
12: **end if**
13: **end if**
14: **end if**
15: Output the initial feasible solutions $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^r\}$.

the new $\mathbf{x}^{\text{best}}$. If no point is better than the original $\mathbf{x}^{\text{best}}$, the best point remains the same. The structure of the direct search is given as below.

The calculation of the matrix J formed by the set of directions (line 3, Algorithm 8) is a key step. When $\mathbf{x}^{\text{best}}$ is not close to the boundary of the feasible domain $\mathcal{S}$, the problem can be considered as an unconstrained one. In this case $J \triangleq [\mathbf{j}^1, \mathbf{j}^2, \ldots, \mathbf{j}^k] = I$, the identity matrix, is a good choice. Otherwise, the set of searching directions $\{\mathbf{j}^1, \ldots, \mathbf{j}^k\}$ should reflect the geometry of any portion of the boundary of the feasible region near $\mathbf{x}^{\text{best}}$.

To calculate J in the latter case, we adopt the idea in [53]. Let $\mathbf{a}^i$ be the i^{th} row of A in (3.3) and b_i be the i^{th} element of $\mathbf{b}$ in (3.3). For a given $\epsilon > 0$, define

$$\mathcal{I} = \{i \mid \mathbf{a}^i \mathbf{x}^{\text{best}} - b_i \geq -\epsilon,\ i = 1, 2, \ldots, m\}, \tag{3.15}$$

Algorithm 8 Local Search $(\mathbf{x}^{\text{best}}, k, \Delta)$

$\mathbf{x}^{\text{best}}$: the current best point.
k: the number of searching directions.
Δ: the stopping parameter.

Set $\phi > 1$ to be the increasing factor.
Let $F(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \in \mathcal{S}, \\ +\infty, & \text{otherwise.} \end{cases}$
Compute the matrix J formed by the set of directions, $J = [\mathbf{j}^1, \ldots, \mathbf{j}^k]$.
for $i = 1$ to k **do**
  if $F(\mathbf{x}^{\text{best}} + \Delta \times \mathbf{j}^i) < F(\mathbf{x}^{\text{best}})$ **then**
    Set $\mathbf{x}^{\text{best}} = \mathbf{x}^{\text{best}} + \Delta \times \mathbf{j}^i$, $Successful\ improvement = true$.
    Set $\Delta = \Delta \times \phi$ (increase the stopping parameter).
    Stop the Local search method and return the new $\mathbf{x}^{\text{best}}$.
  end if
end for
if $F(\mathbf{x}^{\text{best}} + \Delta \times \mathbf{j}^i) \geq F(\mathbf{x}^{\text{best}})$ for all $\mathbf{j}^i$ in J **then**
  Stop the Local search method, set $Successful\ improvement = false$ and return the original $\mathbf{x}^{\text{best}}$.
end if
Output the updated $\mathbf{x}^{\text{best}}$ and updated $Successful\ improvement$.

to be the active set whose elements are the indices of the active constraints. Let $\widetilde{A}$, $\widetilde{\mathbf{b}}$ be the matrix and vector that correspond to the active set:

$$\widetilde{A} = \left[\begin{array}{c} \mathbf{a}^i \end{array} \right] \text{and } \widetilde{\mathbf{b}} = \left[\begin{array}{c} b_i \end{array} \right], \ i \in \mathcal{I}. \tag{3.16}$$

Define

$$\mathbf{v}^i \triangleq \mathbf{v}^i(\mathbf{x}^{\text{best}}, \epsilon) = \mathbf{a}^{i^T}, \ i \in \mathcal{I}. \tag{3.17}$$

Geometrically $\mathbf{v}^i(\mathbf{x}^{\text{best}}, \epsilon)$ is the outward-pointing normal to the corresponding facet of $\mathcal{S}$. Define $K(\mathbf{x}^{\text{best}}, \epsilon)$ to be the cone generated by the vectors in $\{\mathbf{v}^i(\mathbf{x}^{\text{best}}, \epsilon), \ i \in \mathcal{I}\}$, and its polar cone $K^0(\mathbf{x}^{\text{best}}, \epsilon) = \{\mathbf{u} | \mathbf{u}^T\mathbf{v} \leq 0, \ \forall \ \mathbf{v} \in K(\mathbf{x}^{\text{best}}, \epsilon)\}$. Then the search directions can be formed by the vectors which generate the cone $K^0(\mathbf{x}^{\text{best}}, \epsilon)$.

If the vectors $\{\mathbf{v}^i\}$ that generate the cone $K(\mathbf{x}^{\text{best}}, \epsilon^*)$ are linearly independent for some $\epsilon^* > 0$, one can construct the generators of the cone $K^0(\mathbf{x}^{\text{best}}, \epsilon)$ in the following way: let V denote the matrix whose columns are $\{\mathbf{v}^i\}$. Suppose there are s vectors, V is an $n \times s$ matrix. Let N be the matrix whose columns are in the basis of the null space of V^T. Then one can show that for any $\epsilon, \ 0 < \epsilon < \epsilon^*$, a set of generators of $K^0(\mathbf{x}^{\text{best}}, \epsilon)$ can be found among the columns of N, $V(V^TV)^{-1}$ and $-V(V^TV)^{-1}$.

The next task is to determine a set of linearly independent vectors $\{\mathbf{v}^i\}$. If ϵ is set to be too large, there could be more rows in $\widetilde{A}$ and there is a higher probability that the rows are linearly dependent. If it is too small, the directions obtained could be useless. Thus the direct search used in the EM-Lin method tries to dynamically decrease ϵ to achieve our goal. The algorithm is summarized in Algorithm 9.

Algorithm 9 CalcJ($\mathbf{x}^{\text{best}}, \epsilon_{\text{tol}}$)

$\mathbf{x}^{\text{best}}$: the current best point.
$\epsilon_{\text{tol}} > 0$: the stopping tolerance.

Set $\epsilon > 0$ to be the stopping parameter.
Set $0 < \vartheta < 1$ to be the decreasing factor.
while $\epsilon > \epsilon_{\text{tol}}$ **do**
  Let $\widetilde{A}$ be the matrix defined in (3.16).
  if $\widetilde{A}$ is of full row rank and $rank(\widetilde{A}) > 0$ **then**
    Find a QR factorization of $\widetilde{A}^T$: $[Q, R] = \mathsf{qr}(\widetilde{A}^T)$.
    Set $B = Q(R^T)^{-1}$, $N = I - B\widetilde{A}$.
    Set $J = [B \ -B \ N \ -N]$.
  else if $\dim(\widetilde{A}) = 0$ **then**
    Set $J = [I \ -I]$.
  else
    Set $\epsilon = \vartheta \times \epsilon$ (decrease ϵ to make $\widetilde{A}$ full row rank).
  end if
end while
Output the matrix $J = [\mathbf{j}^1, \mathbf{j}^2, \ldots, \mathbf{j}^k]$ where $\{\mathbf{j}^i, \ i = 1, 2, \ldots, k\}$ are the directions used to generate new trial points.

3.4 Calculation of Total Force Vector

The EM method is a population-based iterative method. For such kind of methods, the part that has most influence on its efficiency is the procedure that decides the positions of the points in each iteration.

In the original EM method, the essential part is the calculation of the total force vector *CalcF* and the move procedure *Move*. *CalcF* is kept the same in EM-Lin since it preserves the spirit of the EM method and does not violate the feasibilities of the points. However, the *Move* procedure has to be modified so that it meets the feasibility requirement of the new points. The scheme of calculating the total force is listed in Algorithm 10.

Algorithm 10 CalcF(ν) ($\nu \sim U(0,1)$)

ν: the parameter for the perturbed point.
$\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^r$: the current population.

$\mathbf{x}^p = \text{argmax}\{\|\mathbf{x}^{\text{best}} - \mathbf{x}^i\|,\ i = 1, 2, \ldots, r\}$.
for $i = 1$ to r **do**
  $q_i = \exp\left(-n \times \frac{f(\mathbf{x}^i) - f(\mathbf{x}^{\text{best}})}{\sum_{k=1}^{r}[f(\mathbf{x}^k) - f(\mathbf{x}^{\text{best}})]}\right)$ and set $\mathbf{f}^i = \mathbf{0}$.
end for
for $i = 1$ to r **do**
  for $j = 1$ to r **do**
    if $i \neq j$ **then**
      if $i \neq p$ **then**
        $\mathbf{f}^{ij} = (\mathbf{x}^j - \mathbf{x}^i)\frac{q_i q_j}{\|\mathbf{x}^j - \mathbf{x}^i\|^2}$.
      else
        $\lambda \sim U(0,1)$ and $\mathbf{f}^{ij} = (\mathbf{x}^j - \mathbf{x}^i)\frac{\lambda q_i q_j}{\|\mathbf{x}^j - \mathbf{x}^i\|^2}$.
        if $\lambda < \nu$ **then**
          $\mathbf{f}^{ij} = -\mathbf{f}^{ij}$ (reverse direction).
        end if
      end if
      if $f(\mathbf{x}^j) < f(\mathbf{x}^i)$ **then**
        $\mathbf{f}^i = \mathbf{f}^i + \mathbf{f}^{ij}$ (attraction).
      else
        $\mathbf{f}^i = \mathbf{f}^i - \mathbf{f}^{ij}$ (repulsion).
      end if
    end if
  end for
end for
Output the total force vectors $\{\mathbf{f}^1, \mathbf{f}^2, \ldots, \mathbf{f}^r\}$.

3.5 Movement According to Total Force Vector

In EM-Lin, the mechanism of movement of the points is similar to that of the original EM. But instead of moving inside the feasible region formed only by the lower and upper bounds, the points have to shift inside a polyhedron formed by the bounds and linear constraints. Thus some necessary changes to the parameter RNG in (3.18) are needed. Recall that in Section 3.1, some preprocesses have been done, all the constraints are treated as linear constraints. The feasible region is $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n | \ A\mathbf{x} \leq \mathbf{b}\}$.

To simplify the notation, let $\mathbf{f}^i \leftarrow \frac{\mathbf{f}^i}{\|\mathbf{f}^i\|}$, $(i = 1, 2, \ldots, r \text{ and } i \neq best)$ be the normalized force vector. Recall the formulation in Chapter 2, equation (2.8) is equivalent to the following:

$$\mathbf{x}^i_{\text{new}} = \mathbf{x}^i + \lambda \mathbf{f}^i(RNG_i), \ i = 1, 2, \ldots, r \text{ and } i \neq best, \tag{3.18}$$

where $\lambda \sim U(0, 1]$ and $\mathbf{x}^i_{\text{new}}$ is the updated point.

For a point $\mathbf{x}^i$ in the population, let $\mathcal{L}_i$ be the set of indices corresponding to the constraints which may lead to infeasibility,

$$\mathcal{L}_i = \{j \mid \mathbf{a}^j \mathbf{f}^i > 0, \ j = 1, 2, \ldots, m\}, \tag{3.19}$$

where $\mathbf{a}^j$ is the j^{th} row of matrix A. We can see that for a row vector $\mathbf{a}^k$ such that $\mathbf{a}^k \mathbf{f}^i \leq 0$, no matter how large the step length is, the new point remains feasible. It simply goes further away from the boundary of the feasible region. Therefore, only the rows $\mathbf{a}^j$ whose indices are in $\mathcal{L}_i$ need to be considered. The maximum step length allowed

along the force vector $\mathbf{f}^i$ is given by

$$RNG_i = \begin{cases} \min_{j \in \mathcal{L}_i} \left(\dfrac{b_j - \mathbf{a}^j \mathbf{x}^i}{\mathbf{a}^j \mathbf{f}^i} \right), & \text{if } \mathcal{L}_i \text{ is not empty,} \\ 1, & \text{if } \mathcal{L}_i \text{ is empty.} \end{cases} \tag{3.20}$$

Then, if a direction $\mathbf{f}^i$ is pointing outward to the boundary that may lead the point $\mathbf{x}^i$ to infeasibility, RNG_i will prevent the point from going too far to be out of the feasible domain. Thus equation (3.18) guarantees the feasibility of the updated point.

If a point $\mathbf{x}^i$ is close to the boundary of $\mathcal{S}$ and the direction $\mathbf{f}^i$ exerted on it is pointing outward to that boundary, the RNG_i could be very small. In this case, the point will not be updated to a new position. To resolve this problem, one more step is applied at $\mathbf{x}^i$.

The first thing is to find the active constraints. Let ϵ be a small positive number, $\mathbf{a}^j$ be the j^{th} row of A and b_j be the j^{th} element of $\mathbf{b}$. For a given ϵ, define

$$\mathcal{M}_i = \left\{ j \mid \frac{b_j - \mathbf{a}^j \mathbf{x}^i}{\mathbf{a}^j \mathbf{f}^i} < \epsilon, \quad j = 1, 2, \ldots, m \right\},$$

and

$$A_i = \begin{bmatrix} \mathbf{a}^j \end{bmatrix}, \ j \in \mathcal{M}_i. \tag{3.21}$$

Then, we project the direction $\mathbf{f}^i$ onto the null space of A_i, i.e.,

$$\hat{\mathbf{f}}^i = (I - A_i^T (A_i A_i^T)^{-1} A_i) \mathbf{f}^i. \tag{3.22}$$

Since $A_i \hat{\mathbf{f}}^i = 0$, $\hat{\mathbf{f}}^i$ will not point outward to the active boundary and it is a direction that will not force RNG_i to be very small. Thus, $\hat{\mathbf{f}}^i$ can be used as the new force vector exerted on $\mathbf{x}^i$.

After the points in the population have reached their new locations, two more steps are implemented. First, if those points are too close to the current best point, mark them as inactive and drop them. Second, we seek to avoid redundant function evaluations when two points are too close to each other. These two steps can be easily done by calculating the distances between the corresponding points. They can reduce the number of function evaluations and thus improve the efficiency of the algorithm, especially when dealing with costly function evaluations.

3.6 Termination Criteria

The original EM method stops when the number of iterations exceeds a maximum limit. We keep this as an important criterion, but there are other ways of defining the stopping criteria.

Notice that in the searching procedure there are two cases in which the method could fail to find a better point. The first one is that, before performing the local search procedure, all the updated points are not better than the best point obtained in the previous iteration. The second case is that a failure occurs in the local search step. Notice that Δ is decreased when the two cases happen consecutively. Hence a sufficiently small Δ would indicate that the algorithm could not find better point and it is the time to stop.

In this dissertation the algorithm stops when either the iterations or function evaluations exceed the corresponding limits, or when Δ defined in Algorithm 6 is less than a certain threshold Δ_{tol}.

Chapter 4

Computational Experiments of EM-Lin

Once an algorithm is developed, it is always important and necessary to test its usability and efficiency. In this chapter, we introduce some derivative free optimizers first and compare them with EM-Lin on the aspects of accuracy and efficiency in obtaining global optimal solutions. The performance of EM-Lin is shown by solving a collection of benchmark test problems found in the literature.

4.1 Introduction to Existing Optimizers

The solvers used in our numerical experiments are PSwarm and Genetic Algorithm solver in the MATLAB toolbox.

PSwarm [109] is a hybrid solver for linearly constrained derivative free global optimization. The algorithm uses a coordinate search method which includes a poll step and a search step. The poll step incorporates positive generators for the tangent cone of

the approximated active constraints, and in the search step a particle swarm scheme is utilized for disseminating points in the feasible region.

Genetic Algorithm solver (GA) is a MATLAB toolbox for solving constrained optimization problems. The algorithm is described in [16, 17, 35]. The solver handles linear constraints directly and utilizes the augmented Lagrangian method to handle nonlinear constraints. In our test, only linear constraints are involved. Furthermore, the Pattern Search method is used as a hybrid algorithm for the GA.

All the parameters used in the solvers are set to be the default values. The populations size of each solver is set to be 40. EM-Lin, GA and PSO are all run 10 times. We terminate the iteration using a combination of relative and absolute measures of $f(\mathbf{x})$, i.e., when

$$|f(\mathbf{x}^*) - f_{glob}| \leq \tau_r |f_{glob}| + \tau_a, \tag{4.1}$$

where $f(\mathbf{x}^*)$ is the solution obtained by the algorithm and f_{glob} is the known global optimum. τ_r and τ_a are relative and absolute error tolerances, respectively. In our experiments, we set $\tau_r = 10^{-3}$ and $\tau_a = 10^{-4}$.

When a solver cannot achieve global optimum or the global optimum of the problem is unknown, it stops when it reaches the maximum number of function evaluations which is set to be 10,000 in our experiments.

4.2 Comparisons of EM-Lin and Existing Global Optimizers

In this section, we compare EM-Lin with other global optimization methods for problems with general linear constraints. We collected 73 problems with bounded and linear

constraints from the following resources: Vanderbei [106], CUTEr collection [37], GLOB-ALLib [33], Runarsson and Yao [92], Ji et al [48] and Michalewicz [68, 70]. Among the 73 problems, there are 14 problems whose objective functions are linear functions and 59 problems whose objective functions are nonlinear functions. The dimension of the problems ranges between 2 and 100.

4.2.1 Performances on All Test Problems

Figure 4.1 shows the average number of function evaluations used by EM-Lin to solve the 73 problems.

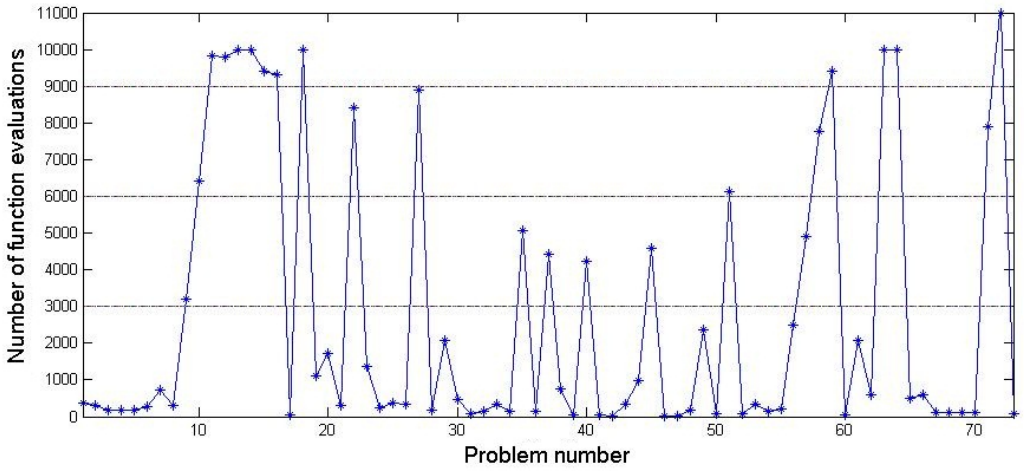

Figure 4.1: Number of function evaluations used by EM-Lin for solving the test problems

Figure 4.2 shows the number of problems solved under different (average) function evaluations.

Figures 4.1 and 4.2 suggest that a large portion of the problems (more than 50) can be solved by EM-Lin in under 2,000 function evaluations. Most problems (more

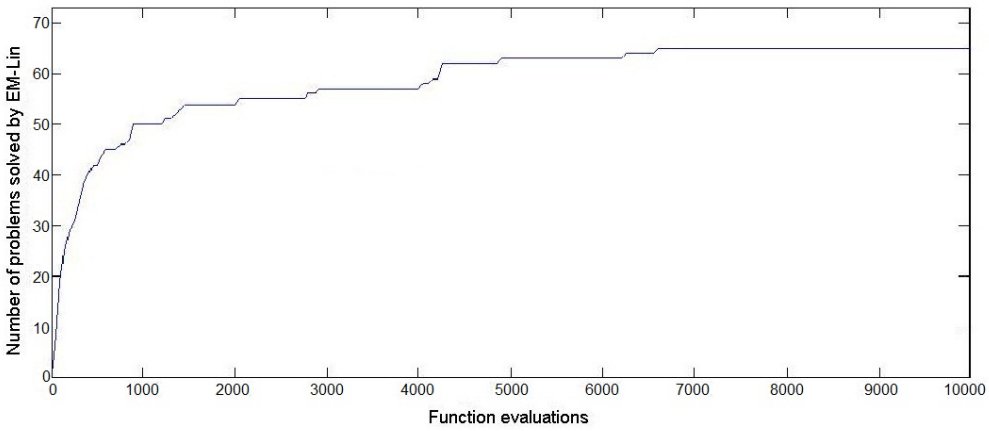

Figure 4.2: Number of problems solved by EM-Lin under different function evaluations

than 60) are solved in under 5,000 function evaluations. The points whose number of function evaluations are 11,000 in Figure 4.1 represent the problems that are not solved optimally in 10,000 function evaluations. (Since we allow the algorithms to finish the current iteration before they stop, the number of function evaluations for some cases could exceed 10,000).

Table 4.1 lists some of the tested problems solved by EM-Lin. The problems have various dimensions and numbers of constraints. It indicates that EM-Lin has the ability to find optimal solutions or near optimal solutions for problems of different sizes. We can also find out that, in general, the number of function evaluations grows when the dimension and/or number of constraints become larger. Furthermore, EM-Lin met some difficulties in solving problem *s224*, which is a linear programming (LP) problem. This fact is true for many LP problems and gives us an area where EM-Lin needs to be improved.

Table 4.1: Problems of various dimensions and number of constraints solved by EM-Lin

Prob	Dim	Cons	Avg. Evals.	Stdev. Evals.	Avg $f(\mathbf{x})$	Best $f(\mathbf{x})$	Stdev. $f(\mathbf{x})$	Known Best
Ji3	2	1	348	31.7007	-5.9948	-5.9955	0.0012	-6.0000
s224	2	2	6115	50.3919	5.7009	0.0000	6.3076	0.0000
hs076	4	3	137	23.1442	-4.6792	-4.6816	0.0016	-4.6818
bunnag5	6	4	298	79.9111	-11	-11	0.0000	-11
avgasa	6	6	373	54.3612	-4.1685	-4.1687	0.0002	-4.1687
s278	6	6	4909	2842.3621	7.8470	7.8434	0.0023	7.8385
bunnag12	20	10	9401	1610.9595	-2782.3868	-4105.2779	1534.9321	-4105.2779
ex2_1_7	20	10	8489	3666.4410	-3688.4141	-4147.5819	637.8759	-4150.4101
goffin	51	50	8892	1308.2944	0.0029	0.0001	0.0017	0.0000
himmelbi	100	12	2088	689.3390	-1754.3000	-1755.0000	0.5516	-1755.0000

To investigate the relationship between dimensionalities of the problems and function evaluations used by EM-Lin, we divide the problems into groups of different dimensions. The median of the problem dimension is 4, and the interval of the dimension is divided into 4 parts: $[1, 2], [3, 4], [5, 10]$ and $[11, 100]$. The number of problems in each interval is $19, 21, 18$ and 15, respectively. Then the average dimensions, average function evaluations and the 95% confidence intervals of the average function evaluations are calculated for the problems in these intervals.

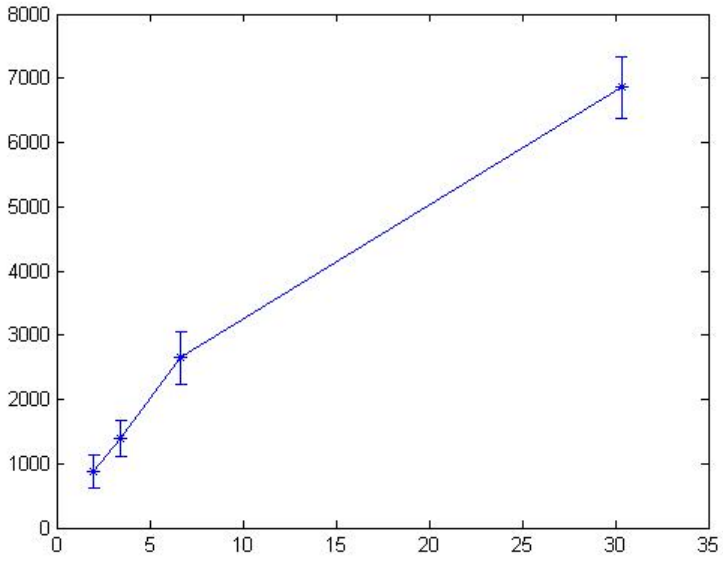

Figure 4.3: Number of function evaluations vs. problem dimensions

The result shown in Figure 4.3 indicates a trend that the number of function evaluations increases as the dimension of the problem grows.

Performance Profile

In the next part we are interested in comparing the existing solvers with EM-Lin. We present the numerical results in the form of *performance profiles*, as described in [25], for evaluating and comparing the performances of optimization softwares. The performance profile for a solver is the (cumulative) distribution function for a performance metric. This procedure was developed to benchmark optimization softwares, i.e., to compare different solvers on several (possibly many) test problems. One advantage of the performance profiles is that the tested solvers can be presented in one figure where each solver has a cumulative distribution function that represents its performance.

Benchmark results are generated by running a set of solvers $\mathbb{S}$ on a set $\mathbb{P}$ of problems and recording information of interests such as the number of function evaluations and the objective function values. We assume that we have n_s solvers and n_p problems.

Firstly, we are interested in using function evaluations as a performance measure; although, the ideas below can be used with other measures. For each problem p and solver s, we define

$$t_{p,s} = \text{the \# of function evaulations required to solve problem } p \text{ by solver } s. \tag{4.2}$$

and performance ratio

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} |\ s \in \mathbb{S}\}} (\geq 1). \tag{4.3}$$

We assume that a parameter $r_M \geq r_{p,s}$, for all p and s, is chosen, and $r_{p,s} = r_M$ if and only if solver s does not solve problem p.

Then we define

$$\rho_s(\tau) = \frac{1}{n_p}(\text{the \# of elements in } \{p \in \mathbb{P} |\ r_{p,s} \leq \tau\}). \tag{4.4}$$

$\rho_s(\tau)$ represents the probability for solver $s \in \mathbb{S}$ whose performance ratio $r_{p,s}$ is within a factor $\tau \geq 1$.

In our study, the term of *performance profile* is kept in use as is in [25]. According to [25], a plot of the performance profile reveals the major performance characteristics. In particular, if the set of problems $\mathbb{P}$ is suitably large and representative for problems that are likely to occur in applications, then the solvers with large probability $\rho_s(\tau)$ are preferred.

Function Evaluations Figures 4.4, 4.6 and 4.7 are the performance profiles for the average, minimum and maximum number of function evaluations for the three tested solvers in 10 runs. There are several points to be addressed in these figures. In each figure, the x-axis is τ, and the y-axis is $\rho_s(\tau)$ defined above. First, the line of EM-Lin is higher than the other two algorithms in all three figures, which means it has the highest probability of being the optimal solver. For instance, in Figure 4.4, EM-Lin solves about 60% problems using least function evaluations, while the percentages for PSO and GA are 23% and 11%, respectively.

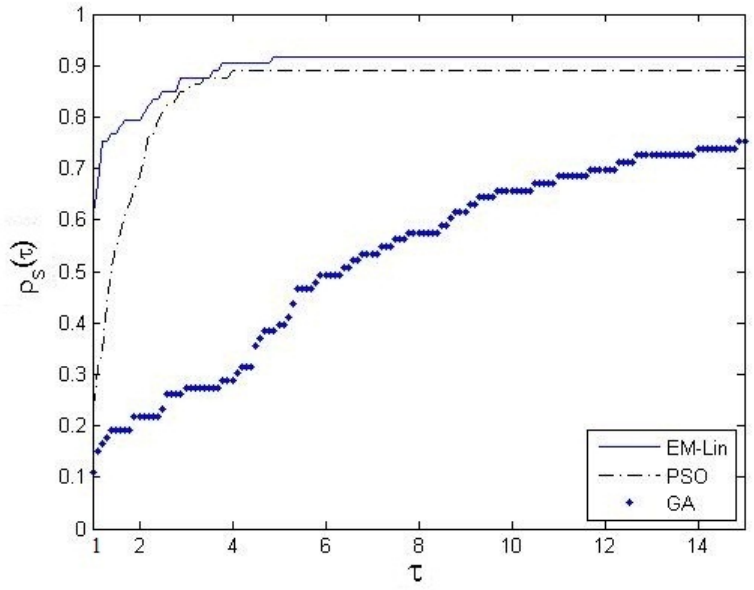

Figure 4.4: Performance profile of the average number of function evaluations on [1, 15]

In Figure 4.4, the line of EM-Lin is higher than the line of PSO for most of the τ, and both of them are higher than the line of GA. Also, EM-Lin is especially competitive for the smaller factors ($\tau \in [1, 3.3]$), which suggests that EM-Lin can solve more problems with relatively small number of function evaluations. It can be seen that EM-Lin solves approximately 75% of the problems with a factor $\tau = 1.2$, which means that in any of these 75% problems, say, problem p, if the smallest number of function evaluations used in all the solvers is t_p^*, then the number of function evaluations used by EM-Lin is no more than $1.2t_P^*$. The corresponding factors for PSO and GA are 2.1 and 14.8, respectively. Similarly, if we are interested in the solver that can solve 80% of the problems with the greatest efficiency, then EM-Lin also stands out.

When τ is large enough, $\rho_s(\tau)$ represents the percentage of problems that a solver s could eventually solve at least once. We set $\tau = r_M = 1,000$, scale the plot and present

Figure 4.5, which is the performance profile for $\log_2(r_{p,s})$. Here, the probability $\rho_s(\tau)$ is defined below:

$$\rho_s(\tau) = \frac{1}{n_p}(\text{the \# of elements in}\{p \in \mathbb{P}|\ \log_2(r_{p,s}) \leq \tau\}). \tag{4.5}$$

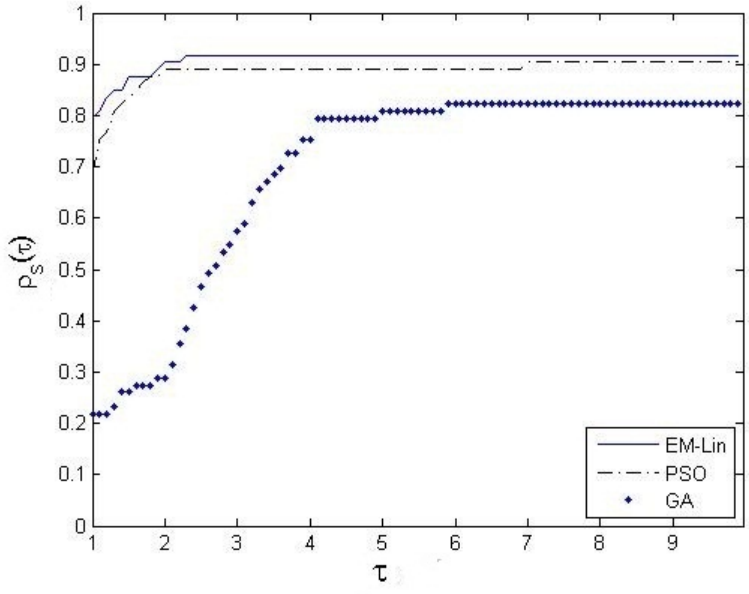

Figure 4.5: Performance profile of the average number of function evaluations on $[1, 2^{10}]$ in $\log_2$ scale

Figure 4.5 indicates that EM-Lin can solve 91.8% of all problems at least once, while PSO and GA can solve 90.4% and 82.2% of all problems at least once. Also, we can see that in the interval $[2, 4]$, the line of GA has a relatively quick increment.

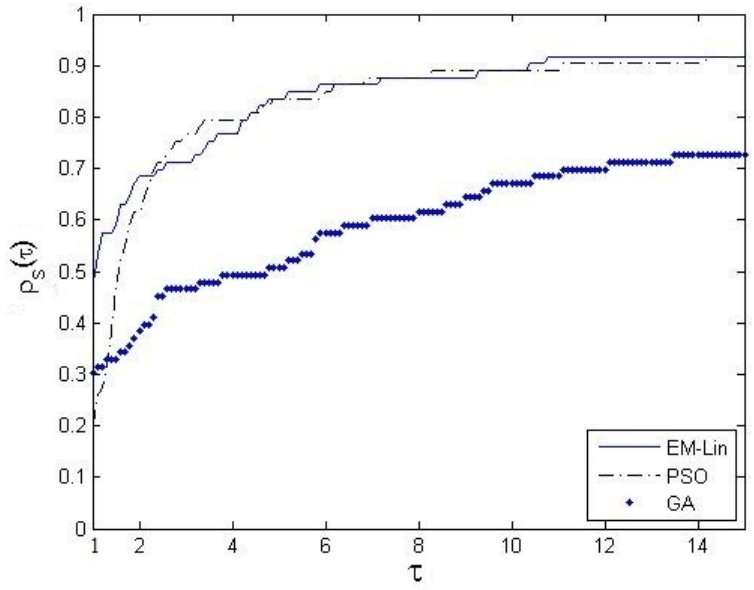

Figure 4.6: Performance profile of the minimum number of function evaluations on [1, 15]

Figure 4.6 shows that, in terms of the minimum number of function evaluations used in 10 runs, 60% of the solutions provided by EM-Lin has the term τ smaller than 1.5, the corresponding percentages for PSO and GA are 50% and 35%. In general, EM-Lin has the best probability $\rho_s(\tau)$ for $\tau \in [1, 2.2]$. PSO has the best probability $\rho_s(\tau)$ for $\tau \in [2.2, 4.1]$. Moreover, both EM-Lin and PSO perform similarly in the interval $[4.1, 15]$.

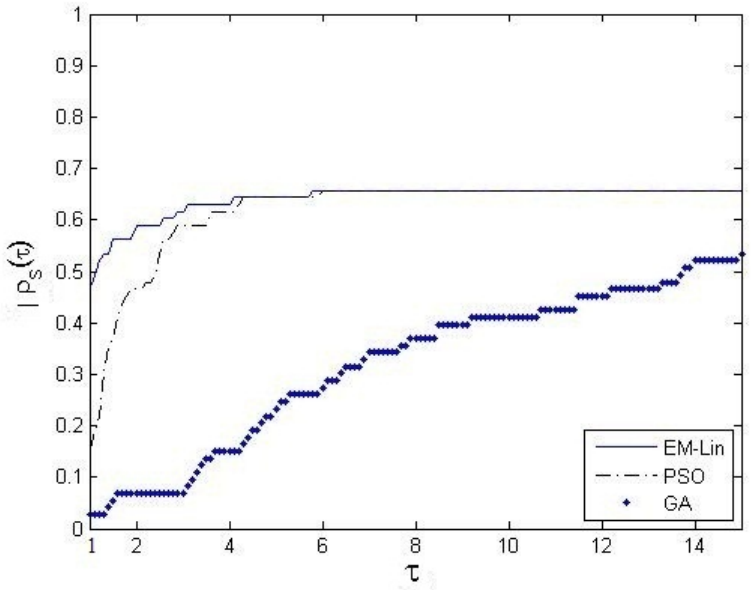

Figure 4.7: Performance profile of the maximum number of function evaluations on [1, 15]

From Figure 4.7, we conclude that EM-Lin is the most efficient solver in terms of the maximum number of function evaluations used in 10 runs. Moreover, it can be seen that EM-Lin can solve 65.8% of all problems in all 10 runs, because the maximum numbers of function evaluations used in these problems are smaller than $10,000$. Similarly, PSO solves 65.8% of the problems and GA solves 64.4% of them.

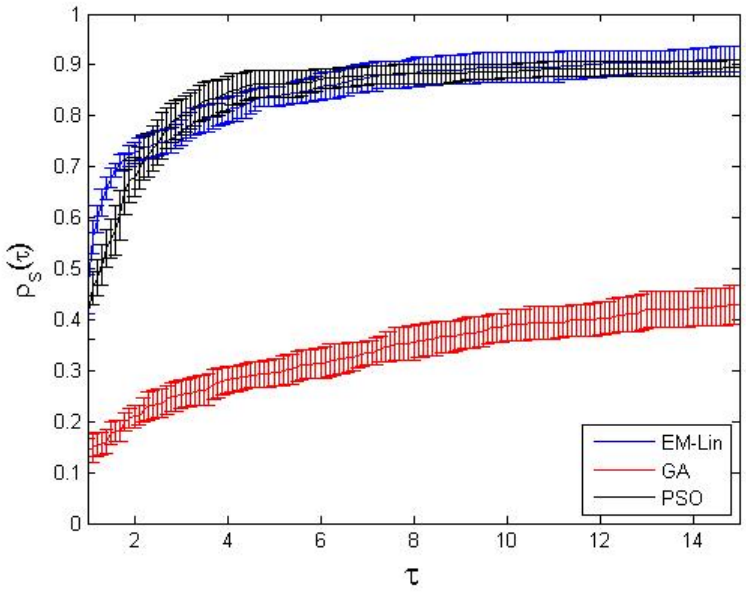

Figure 4.8: Bands of 95% confidence intervals for the performance profile of the number of function evaluations on [1, 15]

Figure 4.8 shows the band of 95% confidence intervals for each solver. We can see that for small τ, the band of EM-Lin is above the band of PSO and they do not overlap, which means that EM-Lin is significantly better than PSO in terms of the number of function evaluations where $\tau \in [1, 2]$. Moreover, the bands of EM-Lin and PSO are above the band of GA for $\tau \in [1, 15]$.

Solution Quality Now we are interested in the objective function values obtained by the tested solvers. Here, the performance ratio is defined as

$$r_{p,s} = \frac{f_{p,s} - f_p^*}{f_p^w - f_p^*} (\leq 1), \tag{4.6}$$

where $f_{p,s}$ is the (average, minimum or maximum) objective value for problem p obtained by solver s, f_p^* and f_p^w are the best and worst objective values for problem p among the solutions of all solvers, respectively. If $f_p^* = f_p^w$, which means that all the solvers have found the same solution, let $r_{p,s} = 0$ for all $s \in \mathbb{S}$. Recall the probability for solver $s \in \mathbb{S}$ that a performance ratio $r_{p,s} \leq \tau$ is defined in (4.4) with $0 \leq \tau \leq 1$ in this situation. For $\tau = 0$, we define

$$\begin{aligned} \rho_s(0) &= \frac{1}{n_p}(\text{the \# of elements in } \{p \in \mathbb{P} |\ r_{p,s} \leq 0\}), \\ &= \frac{1}{n_p}(\text{the \# of elements in } \{p \in \mathbb{P} |\ r_{p,s} = 0\}), \\ &= \frac{1}{n_p}(\text{the \# of problems with } f_{p,s} = f_p^*), \end{aligned} \tag{4.7}$$

and $\rho_s(0)$ is the portion of the problems optimally solved by solver s in 10 runs.

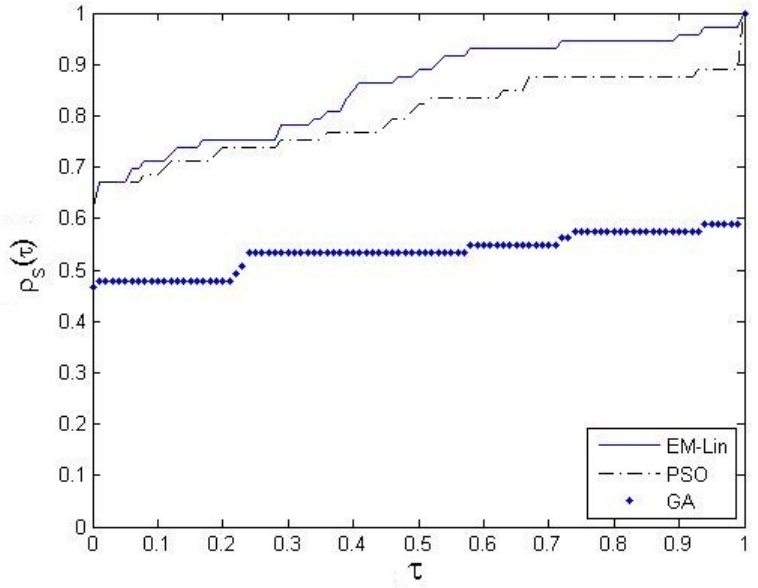

Figure 4.9: Performance profile of the average objective function values

For a fixed $\tau > 0$, a solver with a higher line suggests that it has the ability of providing "$\tau-$good" solutions in more problems. A "$\tau-$good" solution is the one whose performance ratio (4.6) is smaller than or equal to τ.

Figures 4.9, 4.10 and 4.11 are the performance profiles for the average, minimum and maximum solutions obtained by EM-Lin, PSO and GA in their 10 runs.

Figure 4.9 shows that EM-Lin always provides "$\tau-$good" average solutions in more problems than GA and PSO, since the line of EM-Lin is higher than PSO and GA for every τ.

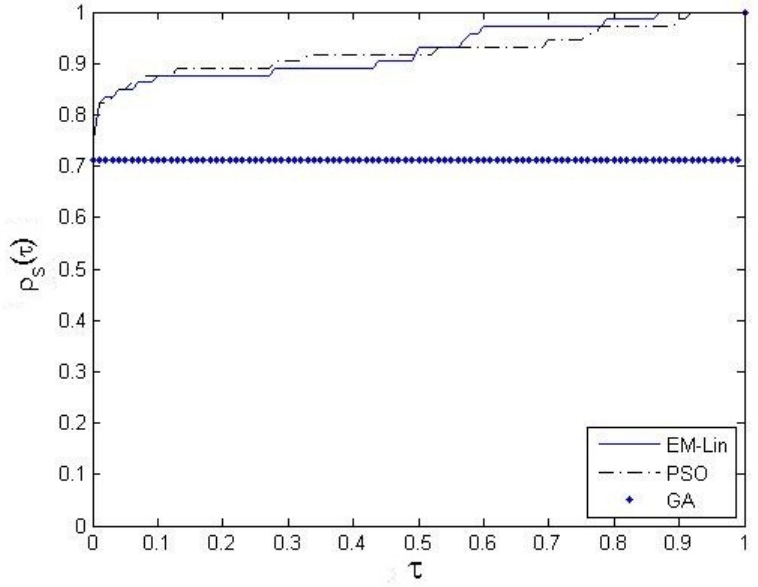

Figure 4.10: Performance profile of the minimum objective function values

Figure 4.10 indicates that, in terms of the minimum objective function values, PSO is better when $\tau \in [0, 0.49]$ and EM-Lin is better for $\tau \in [0.49, 1]$.

From Figure 4.11, we can see that, in terms of the maximum objective function values,

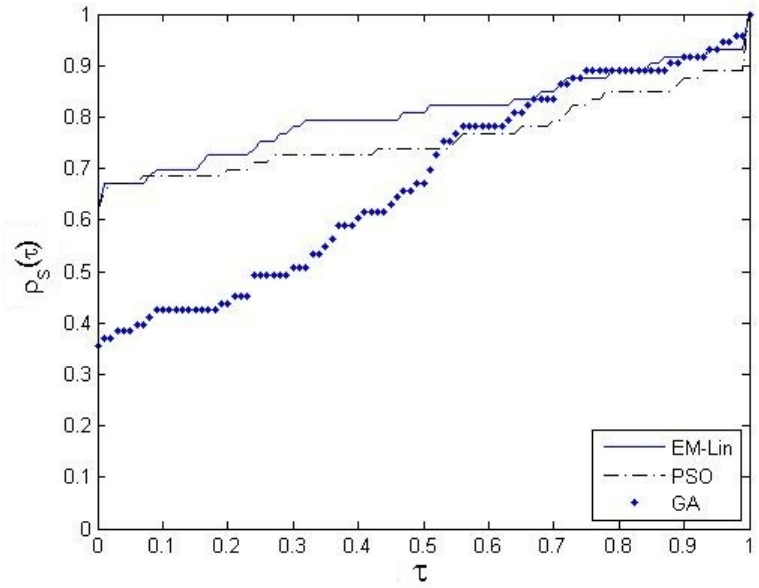

Figure 4.11: Performance profile of the maximum objective function values

EM-Lin is almost always better than PSO, while GA becomes competitive for $\tau \geq 0.52$.

Figure 4.12 shows the band of 95% confidence intervals of each solver in terms of the objective function values. It can be seen that the bands of EM-Lin and PSO overlap heavily, which means that the performances of EM-Lin and PSO are similar in terms of the objective function values.

In summary, if one is interested in the average objective function values, EM-Lin is a good choice, which means that EM-Lin is more stable. While if one seeks for best solutions, both EM-Lin and PSO are very competitive.

4.2.2 Solution Quality on Hard Problems

In this section, we mark the hard problems as those were not solved by a solver s in 4 runs or more out of the 10 runs. There are in total 15 hard problems for EM-Lin, 19 for

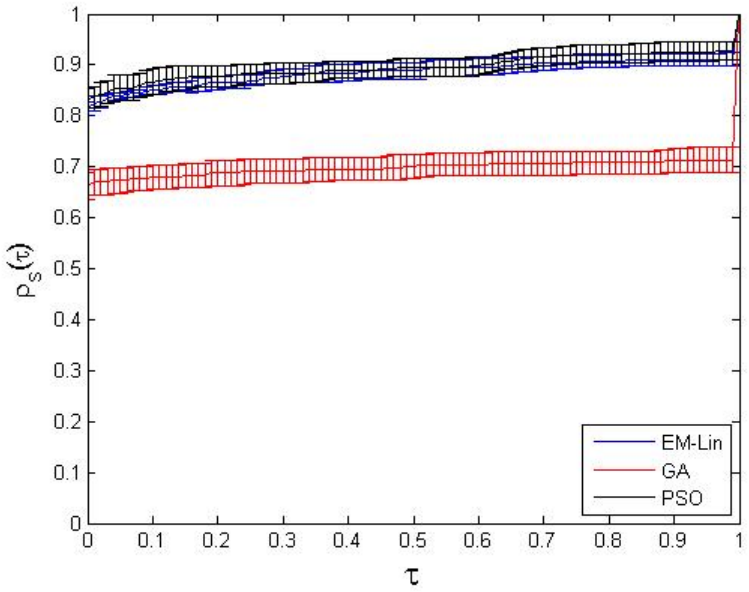

Figure 4.12: Bands of 95% confidence intervals for the performance profile of the objective function values

PSO and 18 for GA. For the hard problems for EM-Lin, we are interested in the best solutions they have ever achieved. The computational results are shown in Table 4.2. In the table, N/A means that the solver did not find a feasible solution.

In Table 4.2, we notice that

1) Among the 15 problems, EM-Lin solved optimally 10 problems at least once: *bunnag7*, *bunnag8*, *bunnag9*, *bunnag12*, *bunnag13*, *ex*2_1_7, *hs044*, *s279*, *s280* and *tfi2*.

2) PSO also solved 10 problems optimally at least once: *bunnag7*, *bunnag8*, *bunnag11*, *bunnag12*, *ex*2_1_7, *hs044*, , *s279*, *s280*, *s359* and *tfi2*.

3) GA solved 4 problems optimally at least once: *bunnag7*, *hs044*, *s279* and *s280*.

4) EM-Lin outperforms PSO in 5 problems (*bunnag9*, *bunnag13*, *ex*2_1_7, *ex*2_1_10 and *s392*) while PSO outperforms EM-Lin in 3 problems (*bunnag10*, *bunnag11* and *s359*).

Table 4.2: Comparison of different methods using 10,000 function evaluations

Problem	n	EM-lin	PSO	GA	Known best
bunnag7	10	-39.0000	-39.0000	-38.9921	-39.0000
bunnag8	20	-394.4029	-394.3840	-160.1755	-394.7506
bunnag9	20	-883.9029	-828.8544	-679.7276	-884.7506
bunnag10	20	-8224.4559	-8315.1801	-3486.4060	-8695.0119
bunnag11	20	-682.5842	-753.9962	-630.1617	-754.7506
bunnag12	20	-4105.2779	-4105.2779	-524.7473	-4105.2779
bunnag13	20	49359.1934	51431.0000	154925.5218	49318.0000
ex2_1_10	20	101183.1904	121737.4909	153480.6965	49318.0180
ex2_1_7	20	-4147.5819	-4146.4153	-378.5011	-4150.4101
hs044	4	-14.9923	-14.9987	-14.9864	-15.0000
s279	8	10.6106	10.6157	10.6168	10.6059
s280	10	13.3886	13.3869	13.3906	13.3754
s359	5	-5.4711E+06	-5.4958E+06	N/A	-5.5045E+06
s392	30	-1.0662E+06	-1.0418E+06	-331242.8649	-1.1012E+6
tfi2	3	0.6496	0.6493	N/A	0.6490

5) None of the solver has found a close-to-optimal solution for the linear programming problem *ex*2_1_10. This shows that their abilities of solving linear programming problems need to be improved.

To quantify the distances of the results given by EM-Lin, PSO and GA from the known best solutions, in Table 4.3, the errors of EM-Lin, PSO and GA are defined:

$$e_{EM} = \frac{|f^*_{EM} - f_{glob}|}{|f_{glob}|}, \tag{4.8}$$

$$e_{PSO} = \frac{|f^*_{PSO} - f_{glob}|}{|f_{glob}|}, \tag{4.9}$$

$$e_{GA} = \frac{|f^*_{GA} - f_{glob}|}{|f_{glob}|}, \tag{4.10}$$

where f^*_{EM}, f^*_{PSO}, f^*_{GA} are the best values obtained by EM-Lin, PSO and GA, respectively. f_{glob} is the known best value. In the table, N/A means GA can not find a feasible solution

for that problem.

Table 4.3: Comparison of errors under the budget of 10,000 objective function evaluations

Problem	e_{EM}	e_{PSO}	e_{GA}
bunnag7	**0.0000**%	**0.0000**%	0.0203%
bunnag8	**0.0881**%	0.0929%	59.4236%
bunnag9	**0.0958**%	6.3177%	23.1730%
bunnag10	5.4118%	**4.3684**%	59.9034%
bunnag11	9.5616%	**0.1000**%	16.5073%
bunnag12	**0.0001**%	0.0002%	87.2177%
bunnag13	**0.0835**%	4.2844%	214.1359%
ex2_1_10	**105.1648**%	146.8418%	211.2061%
ex2_1_7	**0.0681**%	0.0963%	90.8804%
hs044	0.0513%	**0.0087**%	0.0907%
s279	**0.0443**%	0.0924%	0.1028%
s280	0.0987%	**0.0860**%	0.1136%
s359	1.8778%	**0.1574**%	N/A
s392	**3.1784**%	5.3941%	69.9198%
tfi2	0.0924%	**0.0462**%	N/A

Table 4.3 shows that

1) EM-Lin solved 10 problems with error smaller than 0.1%, they are considered as solved optimally by EM-Lin.

2) Other than these 10 problems, there are 2 problems whose errors are between 0.1% and 5%: *s359* and *s392*. There are 2 problems whose errors are between 5% and 10%: *bunnag10* and *bunnag11*.

The bold numbers in Table 4.3 are the smallest errors for the corresponding problems.

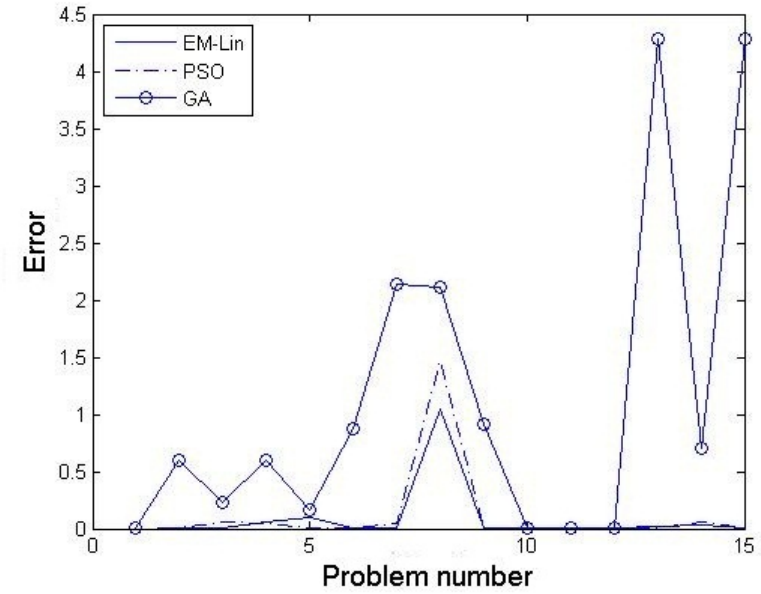

Figure 4.13: Errors of EM-Lin, PSO and GA for hard problems

4.3 Conclusion

In this chapter, the capability of EM-Lin for solving the global optimization problems with linear constraints has been investigated through the numerical experiments on selected test problems. We have tested the EM-Lin algorithm and compared it with GA and PSO. Without using the first or second order information, EM-Lin has achieved better performance than the other two in terms of the number of function evaluations and/or the quality of best solutions. More specifically, EM-Lin is able to achieve highly efficient results for most of the test problems. Moreover, when solving the problems with lower degree of complexity, EM-Lin converges rapidly to the global optimum in terms of the function evaluations.

Chapter 5

Application of EM-Lin to Financial Planning

In this chapter, a real world application of EM-Lin to the long-term financial planning is presented. This financial planning problem is one of the current research topics, and both deterministic algorithm and stochastic algorithms have been applied to solve it. The deterministic algorithm implemented in this chapter is the branch and bound method, whose mechanism will be explained briefly in Section 5.2. The stochastic algorithms used here are EM-Lin, GA and PSO, which have already been introduced in the previous chapters. In the last section of this chapter, the computational results provided by the above 4 algorithms are shown. Some conclusions can be drawn according to these results.

5.1 Problem Description

In [63], a problem in financial planning under uncertainty is introduced, it is the allocation of financial assets to broad investment categories over a long horizon (10 to 20 years).

The long-term asset allocation problem plays a critical role in successful investing. It has been the subject of research over many years.

The mathematical model of the long-term financial planning is formulated as follows. The set of time steps is defined as

$$t \in \{1, 2, \ldots, T\}. \tag{5.1}$$

The time steps are discrete with a unit of one year. We are interested in a long planning horizon of 10 to 20 plus years. Define the number of assets by

$$i \in \{1, 2, \ldots, I\}, \tag{5.2}$$

and the set of scenarios by

$$s \in \{1, 2, \ldots, S\}. \tag{5.3}$$

Five hundred to one thousand scenarios are typically required for adequately capturing the uncertain future trends in realistic portfolio problems.

The following parameters specify the model:

- w_0: initial wealth (in dollars).
- $r_{i,t}^s$: return (in %) for asset i, in time period t, under scenario s, $i = 1, 2, \ldots, I,\ t = 1, 2, \ldots, T$ and $s = 1, 2, \ldots, S$.
- p^s: probability of occurrence for scenario s, $s = 1, 2, \ldots, S$.

The probabilities $\{p^s\}$ have to satisfy the equation $\sum_{s=1}^{S} p^s = 1$.

The decision variables are defined as follows:

w_t^s: wealth (in dollars) at time period t, according to scenario s, $t = 1, 2, \ldots, T$ and $s = 1, 2, \ldots, S$.

λ_i: proportion of wealth invested in asset category i (constant across time), $i = 1, 2, \ldots, I$, $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_I)$.

The first constraint is that all the fractions must be positive and add up to one:

$$\sum_{i=1}^{I} \lambda_i = 1 \text{ and } 0 \leq \lambda_i \leq 1,\ i = 1, 2, \ldots, I. \tag{5.4}$$

Wealth w_1^s at the end of the first period becomes

$$w_1^s = w_0 \sum_{i=1}^{I} (1 + r_{i,1}^s)\lambda_i,\ s = 1, 2, \ldots, S. \tag{5.5}$$

The relationship between w_t^s, w_{t-1}^s and λ_i can be shown as

$$w_t^s = w_{t-1}^s \sum_{i=1}^{I} (1 + r_{i,t}^s)\lambda_i,\ t = 2, 3, \ldots, T,\ s = 1, 2, \ldots, S. \tag{5.6}$$

In this way, we can express the wealth w_T^s for scenario s at the end of the last period T as a function of only λ_i, $i = 1, 2, \ldots, I$, i.e.,

$$w_T^s = w_0 \prod_{t=1}^{T} \left[\sum_{i=1}^{I} (1 + r_{i,t}^s)\lambda_i \right],\ s = 1, 2, \ldots, S. \tag{5.7}$$

Then the expected wealth (mean) at the end of the last period T becomes

$$\text{mean}(w_T) = \sum_{s=1}^{S} p^s w_T^s = w_0 \sum_{s=1}^{S} p^s \left\{ \prod_{t=1}^{T} \left[\sum_{i=1}^{I} (1 + r_{i,t}^s)\lambda_i \right] \right\}. \tag{5.8}$$

The variance $\text{var}(w_T)$ is

$$\text{var}(w_T) = \sum_{s=1}^{S} p^s [w_T^s - \text{mean}(w_T)]^2. \tag{5.9}$$

The goal of the long-term financial planning system is to maximize the weighted total wealth $\text{mean}(w_T)$ subtracting the weighted variance $\text{var}(w_T)$. More specifically, we have

$$\begin{aligned}
\max_{\lambda_i, w_t^s} \quad & \beta \text{mean}(w_T) - (1-\beta)\text{var}(w_T) && (5.10)\\
\text{s.t.} \quad & w_T^s = w_0 \prod_{t=1}^{T} \left[\sum_{i=1}^{I} (1 + r_{i,t}^s)\lambda_i \right], \ s = 1, 2, \ldots, S && (5.11)\\
& \sum_{i=1}^{I} \lambda_i = 1 && (5.12)\\
& 0 \le \lambda_i \le 1, \ i = 1, 2, \ldots, I, && (5.13)
\end{aligned}$$

where β is a weight parameter with $0 \le \beta \le 1$.

Since the equality constraint (5.11) can be eliminated by substituting (5.8), (5.9) and (5.11) into the objective functions, the problem can be reformulated as

$$\begin{aligned}
\min_{\boldsymbol{\lambda}} \quad & f(\boldsymbol{\lambda}) = -\beta w_0 \sum_{s=1}^{S} p^s \left\{ \prod_{t=1}^{T} \left[\sum_{i=1}^{I} (1 + r_{i,t}^s)\lambda_i \right] \right\} \\
& \quad + (1-\beta) w_0^2 \left\{ - \left[\sum_{s=1}^{S} p^s \left[\prod_{t=1}^{T} \left(\sum_{i=1}^{I} (1 + r_{i,t}^s)\lambda_i \right) \right] \right]^2 \right. \\
& \qquad \left. + \sum_{s=1}^{S} p^s \left[\prod_{t=1}^{T} \left[\sum_{i=1}^{I} (1 + r_{i,t}^s)\lambda_i \right] \right]^2 \right\} && (5.14)\\
\text{s.t.} \quad & \sum_{i=1}^{I} \lambda_i = 1 \\
& 0 \le \lambda_i \le 1, \ i = 1, 2, \ldots, I.
\end{aligned}$$

The problem now has a single linear equality constraint and a relatively small number of variables $\lambda_i,\ i = 1, 2, \ldots, I$. But the objective function $f(\boldsymbol{\lambda})$ is complicated because it is a nonconvex multivariate polynomial function of $\boldsymbol{\lambda}$ involving multiple local minima. The existing solvers usually fall into one of the local minima and converge prematurely. Moreover, the evaluation of the objective function is costly since it involves $O(I \times T \times S)$ number of calculations.

5.2 Branch and Bound Method for Solving the Financial Planning Problem

The original work utilized a global optimization method proposed in [64] to solve the above problem. This approach is based on a lower bounding scheme and a successive refinement of converging lower and upper bounds. This method solves not only the financial planning problem, but also other nonconvex optimization problems. However, it needs to use the solutions provided by some convex optimization softwares.

The first step of the branch and bound method is to define the function $F(\boldsymbol{\lambda})$ by augmenting $f(\boldsymbol{\lambda})$ in a certain interval $[\mathbf{l}, \mathbf{u}]$:

$$F(\boldsymbol{\lambda}) = f(\boldsymbol{\lambda}) + \alpha \sum_{i=1}^{I} (l_i - \lambda_i)(u_i - \lambda_i), \tag{5.15}$$

where l_i and $u_i,\ i = 1, 2, \ldots, I$ are the lower and upper bounds of the variables, and

$$\alpha \geq \max \left\{ 0,\ \max_{\lambda_i \in [l_i, u_i]} (-\frac{1}{2} \lambda_i^f) \right\} \tag{5.16}$$

is a nonnegative parameter where $\lambda_i^f,\ i = 1, 2, \ldots, I$ are the eigenvalues of function $f(\boldsymbol{\lambda})$

(please see [64, 63]). The details of calculating α are described in (5.23) — (5.24).

Let $\boldsymbol{\lambda}^f$ and $\boldsymbol{\lambda}^F$ be the eigenvalue vectors of functions $f(\boldsymbol{\lambda})$ and $F(\boldsymbol{\lambda})$, respectively. The eigenvalues in both $\boldsymbol{\lambda}^f$ and $\boldsymbol{\lambda}^F$ are in descending order. There are several properties for $F(\boldsymbol{\lambda})$ and $f(\boldsymbol{\lambda})$ that need to be mentioned. For proof of these properties, please see [64].

Property 1. After being augmented with the term $\alpha \sum_{i=1}^{I}(l_i-\lambda_i)(u_i-\lambda_i)$, $F(\boldsymbol{\lambda})$ becomes a convex function. This is because from (5.15),

$$\lambda_i^F = \lambda_i^f + 2\alpha,\ i = 1, 2, \ldots, I, \tag{5.17}$$

and from (5.16),

$$\lambda_i^f + 2\alpha \geq 0,\ i = 1, 2, \ldots, I. \tag{5.18}$$

Then existing convex optimization solvers can find the global minimal solution of $F(\boldsymbol{\lambda})$.

Property 2. For every $\boldsymbol{\lambda} \in [\mathbf{l}, \mathbf{u}]$, since

$$\alpha \sum_{i=1}^{I}(l_i - \lambda_i)(u_i - \lambda_i) \leq 0, \tag{5.19}$$

we have $F(\boldsymbol{\lambda}) \leq f(\boldsymbol{\lambda})$. And for any $\boldsymbol{\lambda} \in [\mathbf{l}^k, \mathbf{u}^k] \subset [\mathbf{l}, \mathbf{u}]$ $f(\boldsymbol{\lambda})$ is the upper bound of the solution f^* and $F(\boldsymbol{\lambda})$ is a lower bound of f^*, where f^* is the optimal objective function value (5.14).

Property 3. For $\boldsymbol{\lambda} = \mathbf{l}^k$ or $\boldsymbol{\lambda} = \mathbf{u}^k$ with $[\mathbf{l}^k, \mathbf{u}^k] \subseteq [\mathbf{l}, \mathbf{u}]$, $f(\boldsymbol{\lambda}) = F(\boldsymbol{\lambda})$.

Property 4 ([64], *Property 4*). $\max_{\boldsymbol{\lambda}}(f(\boldsymbol{\lambda}) - F(\boldsymbol{\lambda})) = \dfrac{1}{4}\alpha \sum_{i=1}^{I}(u_i - l_i)^2$.

Property 5 ([64], *Property 5*). $\forall \boldsymbol{\lambda} \in [\mathbf{l}^1, \mathbf{u}^1]$, $\forall [\mathbf{l}^2, \mathbf{u}^2]$ such that $[\mathbf{l}^1, \mathbf{u}^1] \subseteq [\mathbf{l}^2, \mathbf{u}^2]$, let

$$F(\boldsymbol{\lambda}, \alpha_j, \mathbf{l}^j, \mathbf{u}^j) \triangleq f(\boldsymbol{\lambda}) + \alpha_j \sum_{i=1}^{I} (l_i^j - \lambda_i)(u_i^j - \lambda_i), \ j = 1, 2, \tag{5.20}$$

where α_1 and α_2 are defined as

$$\alpha_j \geq \max \left\{ 0, \ \max_{\lambda_i \in [l_i^j, u_i^j]} (-\frac{1}{2} \lambda_i^f) \right\}, \ j = 1, 2. \tag{5.21}$$

Then

$$F(\boldsymbol{\lambda}, \alpha_1, \mathbf{l}^1, \mathbf{u}^1) \geq F(\boldsymbol{\lambda}, \alpha_2, \mathbf{l}^2, \mathbf{u}^2). \tag{5.22}$$

Based on the properties, a global optimization algorithm locates the global minimum f^* by constructing converging lower and upper bounds on f^*. From *properties 1* and *2*, $F(\boldsymbol{\lambda})$ is a convex function and is a lower bound of $f(\boldsymbol{\lambda})$. Therefore, its single global minimum within some box constraints is a valid lower bound (denoted as f^L) of the global minimum solution f^* and can be guaranteed to be found with available local optimization algorithms. An upper bound on f^*, denoted as f^U, is then the value of $f(\boldsymbol{\lambda})$ at the global minimum point of $F(\boldsymbol{\lambda})$.

The next step, after establishing an upper and a lower bound on f^*, is to refine them by using *Property 5*. This property implies that the value of $F(\boldsymbol{\lambda})$ at every point, and therefore at the global minimum, is increased by restricting the box in which it has been defined. Define the current box in the first iteration to be the initial rectangle. At each iteration, we choose the current box as the sub-rectangle of the initial rectangle that contains the minimum of the minima of $F(\boldsymbol{\lambda})$ over all sub-rectangles. We can achieve tighter box constraints by halving the current box along its longest side to generate two new sub-rectangles. The lower bound of f^* is then the minimum over all the minima of

$F(\boldsymbol{\lambda})$ in every sub-rectangle composing the initial box, including the two newly generated sub-rectangles. According to *Property 5*, this procedure generates a nondecreasing sequence for f^L.

Furthermore, we construct a nonincreasing sequence for the upper bound f^U by selecting it to be the minimum over all previously recorded upper bounds. Clearly, if the global minimum of $F(\boldsymbol{\lambda})$ in any sub-rectangle is greater than the current upper bound f^U, we can ignore this sub-rectangle because the global minimum of $f(\boldsymbol{\lambda})$ cannot be situated inside it (fathoming step). The basic steps for the branch and bound method are shown in Algorithm 11.

The algorithm is said to be of global convergence, but there is a problem in approximating

$$\alpha \geq \max\left\{0, \max_{\lambda_i \in [l_i, u_i]} (-\frac{1}{2}\lambda_i^f)\right\}, \tag{5.23}$$

in the augmented function $F(\boldsymbol{\lambda})$. Explicit expressions for the eigenvalues λ_i^f as functions of λ_i, in general, cannot be derived. In [64] α is obtained by solving the following problem:

$$\begin{aligned} \min \quad & \lambda^f \\ \text{s.t.} \quad & \det(H(f(\boldsymbol{\lambda})) - \lambda^f I) = 0, \end{aligned} \tag{5.24}$$

where $H(f(\boldsymbol{\lambda}))$ is the Hessian matrix of $f(\boldsymbol{\lambda})$. The equality constraint in the above formulation involves I solutions for λ^f. Each solution $\lambda^f(\boldsymbol{\lambda})$ must then be minimized over $\mathbf{l} \leq \boldsymbol{\lambda} \leq \mathbf{u}$, and the negative one half of this solution is the parameter α.

However, the above problem is not a convex programming problem, so its global minimum solution cannot be found for sure. Therefore the approximation may not be accurate. Besides, it is complicated to solve the above problem (even takes longer time than the convex programming subproblems). In our branch and bound code which is

Algorithm 11 Branch and Bound Method

Initialization.
Select a convergence tolerance $\epsilon > 0$.
$iter = 1$. Set current box to be the initial rectangle.
Initialize f^L and f^U.
Choose a feasible point as the current point $\boldsymbol{\lambda}^c$.
Set $\mathcal{I} = \emptyset$.
while $(f^U - f^L) \geq \epsilon$ **do**
  Update upper bound, $f^U = \min\{f^U, f(\boldsymbol{\lambda}^c)\}$.
  Halve the current box.
  Solve the convex programming problems in both sub-rectangles:

$$\min_{\boldsymbol{\lambda}} F(\boldsymbol{\lambda}) \text{ such that } \sum_{i=1}^{I} \lambda_i = 1, \ \boldsymbol{\lambda} \text{ in } i^{\text{th}} \text{ sub-rectangle } i = 1, 2,$$

  with optimal solutions $\boldsymbol{\lambda}^r_{iter}, \ i = 1, 2$.
  if $F(\boldsymbol{\lambda}^r_{iter}) < f^U$ **then**
    Store $\{\boldsymbol{\lambda}^r_{iter}, F(\boldsymbol{\lambda}^r_{iter})\}$ in $\mathcal{I}$.
  end if
  $iter = iter + 1$.
  Update lower bound, $f^L = F(\boldsymbol{\lambda}^{r'}_{iter'})$, where

$$\boldsymbol{\lambda}^{r'}_{iter'} = \text{argmin}\{F(\boldsymbol{\lambda}^r_{iter}) \mid \{\boldsymbol{\lambda}^r_{iter}, F(\boldsymbol{\lambda}^r_{iter})\} \in \mathcal{I}\}.$$

  Update current point, $\boldsymbol{\lambda}^c = \boldsymbol{\lambda}^{r'}_{iter'}$.
  Update current box as the one that contains $\boldsymbol{\lambda}^c$.
  Delete $\{\boldsymbol{\lambda}^{r'}_{iter'}, F(\boldsymbol{\lambda}^{r'}_{iter'})\}$ from $\mathcal{I}$.
  Check for convergence.
  if $(f^U - f^L) < \epsilon$ **then**
    $\epsilon-$convergence has been reached.
    Return $\boldsymbol{\lambda}^* = \boldsymbol{\lambda}^c$ and $f^* = f(\boldsymbol{\lambda}^c)$.
  end if
end while

developed for the experiment, we just set α to be a large number and decrease it to half at each iteration until it reaches the empirically determined value 1. In fact, from [64], the computational experience indicates that even if the value of α is not large enough to maintain convexity of $F(\boldsymbol{\lambda})$ over the entire box region, the employed local optimization algorithm will most likely converge to the global minimum of the nearly convex function $F(\boldsymbol{\lambda})$.

5.3 EM-Lin for Solving the Financial Planning Problem

In this section, a set of test problems are generated and the computational results are reported. The methods implemented here include the branch and bound method, EM-Lin, GA and PSO. The necessary transformations of the problem is made in Section 5.3.1. The design of test problems are presented in Section 5.3.2, while the computational results are compared and analyzed in Section 5.3.3.

5.3.1 Preprocessing for EM-Lin

In order to apply EM-Lin to solve the financial planning problem, the first step is to transfer the equality constraints

$$\sum_{i=1}^{I} \lambda_i = 1, \tag{5.25}$$

to inequality constraints. It can be done by representing λ_I with $\lambda_I = 1 - \sum_{i=1}^{I-1} \lambda_i$. Then, by using the bound constraint $0 \leq \lambda_I \leq 1$, an inequality constraint is formulated as

$$0 \leq 1 - \sum_{i=1}^{I-1} \lambda_i \leq 1. \tag{5.26}$$

Since $\lambda_i \geq 0,\ i = 1, 2, \ldots, I-1$, the right hand side of (5.26) is unnecessary. The only constraint is

$$\sum_{i=1}^{I-1} \lambda_i \leq 1. \tag{5.27}$$

Note that the objective function has to be changed correspondingly:

$$\overline{f}(\lambda_1, \lambda_2, \ldots, \lambda_{I-1}) = f(\lambda_1, \lambda_2, \ldots, \lambda_{I-1}, 1 - \sum_{i=1}^{I-1} \lambda_i). \tag{5.28}$$

Then problem (5.14) becomes

$$\begin{aligned} \min \quad & \overline{f}(\overline{\boldsymbol{\lambda}}) && (5.29) \\ \text{s.t.} \quad & \sum_{i=1}^{I-1} \lambda_i \leq 1 \\ & \lambda_i \geq 0,\ i = 1, 2, \ldots, I-1, && (5.30) \end{aligned}$$

where $\overline{\boldsymbol{\lambda}} = (\lambda_1, \lambda_2, \ldots, \lambda_{I-1})$.

5.3.2 Design of the Test Problems

According to model (5.14), the objective function represents the expected return (mean(w_T)) subtracts the associated total risk (var(w_T)). The parameters of the problems are decided as follows:

The initial wealth $w_0 = 1$.

The number of assets $I = 9$.

The number of time steps $T = 20$.

The number of scenarios $S = 100$.

The probabilities of the occurrence for scenarios $s = 1, 2, \ldots, S$ are set to be equal at anytime, that is,

$$p^s = \frac{1}{S} = \frac{1}{100},\ s = 1, 2, \ldots, S. \tag{5.31}$$

The returns $r_{i,t}^s$ are uniformly distributed in the range of [-0.07, 0.10] as suggested in [63]. The total number of returns is $I \times T \times S = 9 \times 20 \times 100 = 18000$. Also, we provide some basic information of the sample return rates:

Mean$(r_{i,t}^s) = 0.0150$.

Standard deviation$(r_{i,t}^s) = 0.0462$.

The weight parameter β grows from 0.1 to 1.0 by 0.1 at each time, which means the importance of profit is getting larger and larger compared with the risk. With each different β's, 10 problems are formulated.

The algorithms are coded in MATLAB R2008b and run on an IBM T60 laptop with Intel(R) CPU @ 1.66GHz and 1.5GB of memory.

5.3.3 Computational Results

The branch and bound method is run to solve the problems with the stopping criterion setting as $(f^{UBD} - f^{LBD}) \leq \epsilon = 0.001$. The solution obtained from the branch and bound

method is defined to be the optimal solution. Because of the stochastic nature of EM-Lin, GA and PSO, we apply them to the above problems and run each case for 5 times. The budget of function evaluations is set to be 500. Of course, owing to the specific stopping criteria used in different solvers, they could stop before reaching the budget. The computation results, which include both of the solution qualities and computational time, are shown below.

Solution Quality

In this section, the computational results obtained by EM-Lin, GA and PSO are studied. First, we intend to compare EM-Lin with the branch and bound method. Set f_a^{EM}, f_b^{EM} and f_w^{EM} to be the average value, best value and worst value obtained by EM-Lin in 5 runs, respectively. The solution of branch and bound method is denoted as f_{BnB} which is considered as the optimal value.

Table 5.1: Computational results: EM-Lin vs. BnB

	BnB	EM-Lin				
β	f_{BnB}	f_a^{EM}	f_b^{EM}	f_w^{EM}	Stdev. f^{EM}	95% Confidence Interval
0.1	-0.4476	-0.4102	-0.4476	-0.3472	0.0401	[-0.4600, -0.3604]
0.2	-0.6577	-0.6173	-0.6577	-0.5596	0.0426	[-0.6702, -0.5644]
0.3	-0.8678	-0.8678	-0.8678	-0.8678	0.0000	[-0.8678, -0.8678]
0.4	-1.0779	-1.0779	-1.0779	-1.0779	0.0000	[-1.0779, -1.0779]
0.5	-1.2880	-1.2874	-1.2880	-1.2854	0.0011	[-1.2888, -1.2860]
0.6	-1.4980	-1.4803	-1.4980	-1.4093	0.0397	[-1.5295, -1.4310]
0.7	-1.7082	-1.7082	-1.7082	-1.7082	0.0000	[-1.7082, -1.7082]
0.8	-1.9183	-1.9183	-1.9183	-1.9183	0.0000	[-1.9183, -1.9183]
0.9	-2.1284	-2.1284	-2.1284	-2.1284	0.0000	[-2.1284, -2.1284]
1.0	-2.3384	-2.3384	-2.3384	-2.3384	0.0000	[-2.3384, -2.3384]

It can be seen from Table 5.1 that $f_b^{EM} = f_{BnB}$ in all cases. This means that, in each

case, EM-Lin achieves the optimal value at least once. Moreover, $f_a^{EM} = f_{BnB}$ in 6 cases when $\beta = 0.3,\ 0.4,\ 0.7,\ 0.8,\ 0.9$ and 1.0. It can be concluded that for these 6 cases EM-Lin reaches the optimal solution in all 5 runs. In the cases when $\beta = 0.1,\ 0.2,\ 0.5$ and 0.6, the performances of EM-Lin will be further discussed in the next several paragraphs. The standard deviations and 95% confidence intervals of the results are also shown in Table 5.1.

Let the percentage errors be defined as:

$$e_a^{EM} = \frac{|f_a^{EM} - f_{BnB}|}{|f_{BnB}|}, \tag{5.32}$$

$$e_b^{EM} = \frac{|f_b^{EM} - f_{BnB}|}{|f_{BnB}|}, \tag{5.33}$$

$$e_w^{EM} = \frac{|f_w^{EM} - f_{BnB}|}{|f_{BnB}|}, \tag{5.34}$$

when $f_{BnB} \neq 0$.

Table 5.2: Errors of EM-Lin

β	e_a^{EM}	e_b^{EM}	e_w^{EM}	Stdev. e_a^{EM}	95% Confidence Interval
0.1	8.35%	0.00%	22.43%	0.0897	$[-2.7763\%, 19.4876\%]$
0.2	6.14%	0.00%	14.91%	0.0648	$[-1.9037\%, 14.1889\%]$
0.3	0.00%	0.00%	0.00%	0.0000	$[0.0000\%, 0.0000\%]$
0.4	0.00%	0.00%	0.00%	0.0000	$[0.0000\%, 0.0000\%]$
0.5	0.04%	0.00%	0.20%	0.0009	$[-0.0616\%, 0.1547\%]$
0.6	1.19%	0.00%	5.93%	0.0265	$[-2.1032\%, 4.4717\%]$
0.7	0.00%	0.00%	0.00%	0.0000	$[0.0000\%, 0.0000\%]$
0.8	0.00%	0.00%	0.00%	0.0000	$[0.0000\%, 0.0000\%]$
0.9	0.00%	0.00%	0.00%	0.0000	$[0.0000\%, 0.0000\%]$
1.0	0.00%	0.00%	0.00%	0.0000	$[0.0000\%, 0.0000\%]$

Table 5.2 and Figure 5.1 show the errors of EM-Lin in solving the problems. According

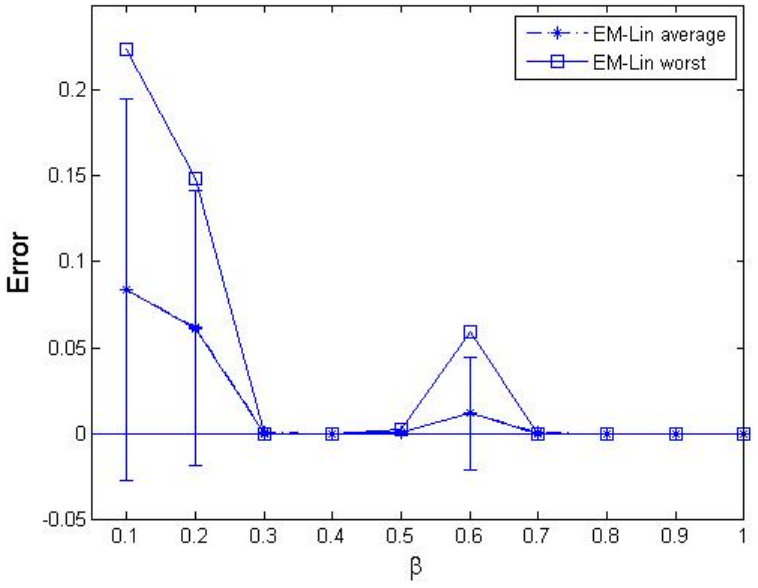

Figure 5.1: Errors of EM-Lin

to Table 5.2 and Figure 5.1, there are 4 problems in which e_a^{EM} is not equal to 0. They are the problems with $\beta = 0.1,\ 0.2,\ 0.5$ and 0.6. For the case with $\beta = 0.5$, both e_a^{EM} and e_w^{EM} are smaller than 1%. When $\beta = 0.6$, e_a^{EM} is smaller than 5%. And for all the cases, the values of e_a^{EM} are lower than 10%, which is relatively small. This shows that the average values provided by EM-Lin are quite promising in all cases. Moreover, the 95% confidence intervals of the errors are shown in in Table 5.2 and Figure 5.1.

Next, we intend to compare the performances of EM-Lin, GA and PSO. Tables 5.3 and 5.4 summarize the computational results given by the branch and bound method, EM-Lin, GA and PSO. Note that the 6 values f_a^{GA}, f_b^{GA}, f_w^{GA}, f_a^{PSO}, f_b^{PSO} and f_w^{PSO} are defined similarly to the values defined for EM-Lin (e.g., f_a^{GA} is the average objective value obtained by GA).

Table 5.3 shows that, all the 95% confidence intervals of $f_a^{EM} - f_a^{GA}$ are to the left of

Table 5.3: Objective function values: EM-Lin vs. other existing optimizers

	EM-Lin vs. GA		EM-Lin vs. PSO	
β	f_a^{EM-GA}	95% Confidence Interval	f_a^{EM-PSO}	95% Confidence Interval
0.1	-0.0544	[-0.0976, -0.0112]	-0.0391	[-0.0822, 0.0040]
0.2	-0.0932	[-0.1441, -0.0423]	0.0006	[-0.0266, 0.0278]
0.3	-0.1117	[-0.1513, -0.0721]	-0.0562	[-0.1049, -0.0075]
0.4	-0.1106	[-0.1600, -0.0612]	-0.0425	[-0.0783, -0.0067]
0.5	-0.1088	[-0.1327, -0.0849]	-0.0167	[-0.0632, 0.0298]
0.6	-0.0863	[-0.1383, -0.0342]	0.0178	[-0.0314, 0.0671]
0.7	-0.0889	[-0.1394, -0.0384]	0.0000	[0.0000, 0.0000]
0.8	-0.0802	[-0.1206, -0.0398]	0.0000	[0.0000, 0.0000]
0.9	-0.0958	[-0.1200, -0.0716]	-0.0002	[-0.0006, 0.0002]
1.0	-0.0677	[-0.1146, -0.0208]	0.0000	[0.0000, 0.0000]

0, which means that EM-Lin is significantly better than GA in all the cases. And there are 2 cases ($\beta = 0.3$ and 0.4) in which EM-Lin are significantly better than PSO. No evidence shows that PSO is significantly better than EM-Lin in any case.

Table 5.4 shows that, comparing with f_a^{GA} and f_a^{PSO}, f_a^{EM} has the lowest value in all cases except for two cases with $\beta = 0.6$ and 0.2. Similarly, comparing with f_w^{GA} and f_w^{PSO}, f_w^{EM} has the lowest value in 7 cases, including the ones with $\beta = 0.3,\ 0.4,\ 0.5,\ 0.7,\ 0.8,\ 0.9$ and 1.0.

Table 5.4: Computational results of EM-Lin, GA, PSO and BnB

	BnB	EM-Lin			GA			PSO		
β	f_{BnB}	f_a^{EM}	f_b^{EM}	f_w^{EM}	f_a^{GA}	f_b^{GA}	f_w^{GA}	f_a^{PSO}	f_b^{PSO}	f_w^{PSO}
0.1	-0.4476	-0.4102	-0.4476	-0.3472	-0.3558	-0.4477	-0.2939	-0.3711	-0.4190	-0.3591
0.2	-0.6577	-0.6173	-0.6577	-0.5596	-0.5241	-0.5456	-0.4920	-0.6179	-0.6477	-0.5871
0.3	-0.8678	-0.8678	-0.8678	-0.8678	-0.7561	-0.8104	-0.7269	-0.8116	-0.8678	-0.7602
0.4	-1.0779	-1.0779	-1.0779	-1.0779	-0.9673	-1.0228	-0.9185	-1.0354	-1.0779	-1.0061
0.5	-1.2880	-1.2874	-1.2880	-1.2854	-1.1786	-1.1942	-1.1469	-1.2707	-1.2880	-1.2017
0.6	-1.4980	-1.4803	-1.4980	-1.4093	-1.3940	-1.4481	-1.3597	-1.4981	-1.4981	-1.4981
0.7	-1.7082	-1.7082	-1.7082	-1.7082	-1.6193	-1.6643	-1.5710	-1.7082	-1.7082	-1.7082
0.8	-1.9183	-1.9183	-1.9183	-1.9183	-1.8381	-1.8774	-1.7940	-1.9183	-1.9183	-1.9183
0.9	-2.1284	-2.1284	-2.1284	-2.1284	-2.0326	-2.0642	-2.0156	-2.1282	-2.1284	-2.1276
1.0	-2.3384	-2.3384	-2.3384	-2.3384	-2.2707	-2.3100	-2.2233	-2.3384	-2.3384	-2.3384

To show the comparison of EM-Lin, GA and PSO more clearly, we pull out the average values obtained by these algorithms and define the percentage errors of average value:

$$e_a^{GA} = \frac{|f_a^{GA} - f_{BnB}|}{|f_{BnB}|}, \tag{5.35}$$

$$e_a^{PSO} = \frac{|f_a^{PSO} - f_{BnB}|}{|f_{BnB}|}, \tag{5.36}$$

when $f_{BnB} \neq 0$ and the percentage error of average value for EM-Lin is defined in (5.32).

Table 5.5: Average errors of the three tested solvers

β	EM-Lin	GA	PSO
0.1	**8.35**%	20.52%	17.09%
0.2	6.14%	20.32%	**6.05**%
0.3	**0.00**%	12.87%	6.48%
0.4	**0.00**%	10.26%	3.94%
0.5	**0.04**%	8.49%	1.34%
0.6	1.19%	6.95%	**0.00**%
0.7	**0.00**%	5.20%	**0.00**%
0.8	**0.00**%	4.18%	**0.00**%
0.9	**0.00**%	4.50%	0.01%
1.0	**0.00**%	2.89%	**0.00**%

Tables 5.5, 5.6 and Figures 5.2 and 5.3 are the comparisons of e_a^{EM}, e_a^{GA} and e_a^{PSO}. We can see that for cases with $\beta = 0.1$ and 0.2, all the solvers have some errors larger than 5%. When β is 0.3 or 0.4, the error of EM-Lin drops quickly to zero and the errors of the other two solvers decrease but are still much larger than zero. When β is 0.5 or higher, errors of EM-Lin and PSO are almost all zero, while the ones of GA are not zero. In summary, in terms of the average objective function values, EM-Lin achieves the best performance in 8 cases out of 10. Table 5.6 provides a conclusion that is similar to the one provided by Table 5.3.

Table 5.6: Errors of the average values: EM-Lin vs. other existing optimizers

	EM-Lin vs. GA		EM-Lin vs. PSO	
β	e_a^{EM-GA}	95% Confidence Interval	e_a^{EM-PSO}	95% Confidence Interval
0.1	-12.1537%	$[-21.8154\%, -2.4921\%]$	-8.7355%	$[-18.3756\%, 0.9047\%]$
0.2	-14.1706%	$[-21.9044\%, -6.4368\%]$	0.0912%	$[-4.0421\%, 4.2246\%]$
0.3	-12.8716%	$[-17.4380\%, -8.3053\%]$	-6.4761%	$[-12.0911\%, -0.8612\%]$
0.4	-10.2607%	$[-14.8404\%, -5.6809\%]$	-3.9429%	$[-7.2621\%, -0.6236\%]$
0.5	-8.4472%	$[-10.3049\%, -6.5895\%]$	-1.2966%	$[-4.9067\%, 2.3135\%]$
0.6	-5.7583%	$[-9.2328\%, -2.2839\%]$	1.1909%	$[-2.0965\%, 4.4784\%]$
0.7	-5.2043%	$[-8.1581\%, -2.2505\%]$	0.0000%	$[0.0000\%, 0.0000\%]$
0.8	-4.1808%	$[-6.2885\%, -2.0731\%]$	0.0000%	$[0.0000\%, 0.0000\%]$
0.9	-4.5010%	$[-5.6369\%, -3.3652\%]$	-0.0094%	$[-0.0294\%, 0.0106\%]$
1.0	-2.8951%	$[-4.9019\%, -0.8884\%]$	0.0000%	$[0.0000\%, 0.0000\%]$

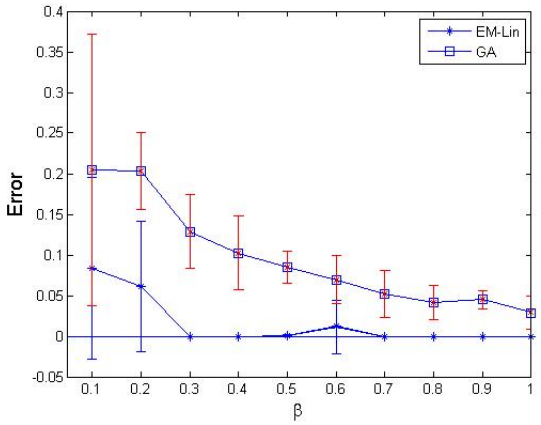

Figure 5.2: Errors of the average values: EM-Lin vs. GA

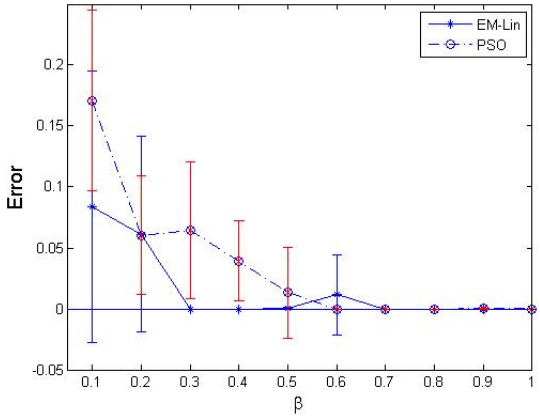

Figure 5.3: Errors of the average values: EM-Lin vs. PSO

From Figure 5.2, we can see that the line of EM-Lin is always below the line of GA. The 95% confidence intervals of EM-Lin are always below those of GA except for the case $\beta = 0.1$.

From Figure 5.3, an observation can be made that the line of EM-Lin is always below the line of PSO except for the cases that $\beta = 0.2$ and 0.6. The 95% confidence intervals of EM-Lin's error always include 0, and the 95% confidence intervals of PSO's error do

not include 0 for $\beta = 0.1 - 0.4$.

Computational Time

Tables 5.7 and 5.8 show the average time (T_a), the least time (T_b) and the most time (T_w) used by these heuristic solvers to solve each case of the problem. Also it shows the time used by the branch and bound method (T_{BnB}).

Table 5.7: Computational time (in seconds) of EM-Lin and branch and bound method

	BnB	EM-Lin		
β	T_{BnB}	T_a^{EM}	T_b^{EM}	T_w^{EM}
0.1	5.1397	0.9727	0.6803	1.7813
0.2	5.1968	1.0287	0.6250	1.4844
0.3	5.1968	0.8138	0.6227	1.5112
0.4	7.8267	0.9801	0.6315	1.5781
0.5	5.1762	0.9038	0.6281	1.1512
0.6	5.3438	1.1328	0.9158	1.6728
0.7	6.2031	0.8388	0.6701	1.0704
0.8	6.0625	0.9664	0.7500	1.0938
0.9	5.6250	1.1939	0.9688	1.5229
1.0	2.3594	1.5189	1.0260	2.0181

Table 5.9 shows that, the times used by EM-Lin are significantly less than those used by GA in all 10 cases according to the 95% confidence intervals. And the times used by PSO are significantly less than those of EM-Lin according to the 95% confidence intervals in 5 cases where $\beta = 0.2, 0.6, 0.8, 0.9$ and 1.

Figures 5.4 and 5.5 plot the ratios of average time for different stochastic solver: $\frac{T_a^{EM}}{T_{BnB}}$, $\frac{T_a^{GA}}{T_{BnB}}$ and $\frac{T_a^{PSO}}{T_{BnB}}$.

From Figure 5.4, we can see that the 95% confidence intervals of EM-Lin are always below those of GA. In Figure 5.3, the 95% confidence intervals of EM-Lin overlap with

Table 5.8: Computational time (in seconds) of GA, PSO and branch and bound method

	BnB	GA			PSO		
β	T_{BnB}	T_a^{GA}	T_b^{GA}	T_w^{GA}	T_a^{PSO}	T_b^{PSO}	T_w^{PSO}
0.1	5.1397	3.9122	2.5286	4.3249	0.6230	0.5278	0.7606
0.2	5.1968	4.2559	3.8308	4.3965	0.5941	0.4957	0.7030
0.3	5.1968	4.2985	4.1493	4.5525	0.5534	0.4942	0.6432
0.4	7.8267	4.2999	4.2112	4.3579	0.5712	0.4539	0.6293
0.5	5.1762	4.3035	4.1010	4.3984	0.6643	0.6136	0.7073
0.6	5.3438	4.2288	4.1835	4.2857	0.6040	0.5685	0.6373
0.7	6.2031	4.2596	4.2267	4.2941	0.6140	0.5651	0.6544
0.8	6.0625	4.2616	4.1121	4.4181	0.7134	0.6602	0.7924
0.9	5.6250	4.3336	4.2828	4.3844	0.6352	0.5728	0.7079
1.0	2.3594	4.3136	4.2445	4.3645	0.6325	0.5756	0.6996

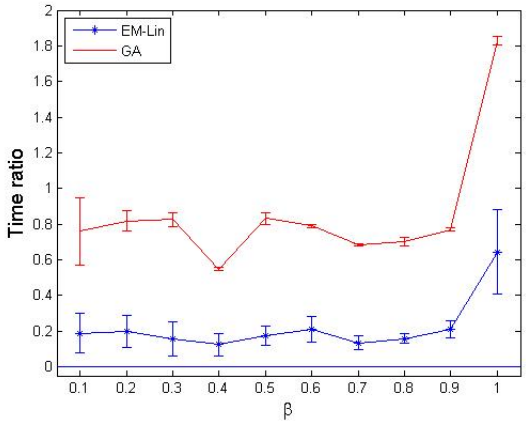

Figure 5.4: Time ratios: EM-Lin vs. GA

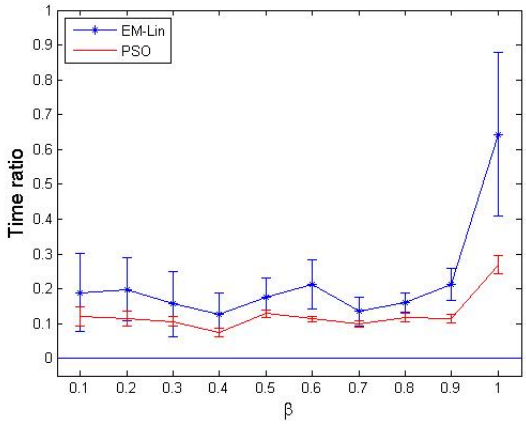

Figure 5.5: Time ratios:EM-Lin vs. PSO

those of PSO for all cases except for the cases that $\beta = 0.6, 0.9$ and 1.0.

By looking at Table 5.9 and Figures 5.4 and 5.5, a conclusion can be made that, the time used by PSO is less than EM-Lin in some cases. The situations are the same for the least time and the most time. This is probably because that in EM-Lin, the procedure of calculating the total force vector is more complicated than the simple procedure used in PSO. EM-Lin uses the more sophisticated scheme in the hope of reducing the iterations

Table 5.9: Computational times (in seconds): EM-Lin vs. other existing optimizers

	EM-Lin vs. GA		EM-Lin vs. PSO	
β	f_a^{EM-GA}	95% Confidence Interval	f_a^{EM-PSO}	95% Confidence Interval
0.1	-2.9395	[-3.8463, -2.0327]	0.3497	[-0.1551, 0.8545]
0.2	-3.2272	[-3.5979, -2.8565]	0.4346	[0.0005, 0.8687]
0.3	-3.4847	[-3.8135, -3.1559]	0.2604	[-0.1626, 0.6834]
0.4	-3.3198	[-3.7624, -2.8772]	0.4089	[-0.0373, 0.8551]
0.5	-3.3997	[-3.5579, -3.2415]	0.2395	[-0.0431, 0.5221]
0.6	-3.0960	[-3.4338, -2.7582]	0.5288	[0.1725, 0.8851]
0.7	-3.4208	[-3.6428, -3.1988]	0.2248	[-0.0303, 0.4799]
0.8	-3.2952	[-3.3547, -3.2357]	0.2530	[0.1289, 0.3771]
0.9	-3.1397	[-3.3630, -2.9164]	0.5587	[0.3473, 0.7701]
1.0	-2.7947	[-3.3254, -2.2640]	0.8864	[0.3847, 1.3881]

and function evaluations. This is more useful when the evaluating objective function is more costly.

Chapter 6

Electromagnetism-like Method for Convex Quadratically Constrained Problems

The Original EM method has been reviewed in Chapter 2, the EM method for linearly constrained problems (EM-Lin) has been introduced in Chapter 3, and the performance and applications of EM-Lin are studied in Chapters 4 – 5. The experimental results show that EM-Lin is highly competitive vs. Genetic Algorithm and Particle Swarm Optimizer.

The objective of this chapter is to adopt the essential parts of the EM method and develop an algorithm for solving optimization problems with more general constraints. Encouraged by the results obtained from previous chapters, we take a further step and propose the EM method for solving the convex quadratically constrained problems (EM-CQ) in this chapter. Because of the increased complexity of the target problems, some modifications are made in EM-CQ. The details are shown in later sections.

6.1 Outline of EM-CQ

We address the convex quadratically constrained optimization problem in the following form:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0,\ i = 1, 2, \ldots, I \\ & \mathbf{x} \in \Omega, \end{aligned} \tag{6.1}$$

where $\Omega = \{\mathbf{x} \in \mathbb{R}^n |\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\},\ \mathbf{l}, \mathbf{u} \in \mathbb{R}^n$, and

$$g_i(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T H^i \mathbf{x} + {\mathbf{h}^i}^T \mathbf{x} + p_i,\ i = 1, 2, \ldots, I, \tag{6.2}$$

with $H^i \succeq 0,\ H^i \in \mathbb{R}^{n \times n},\ \mathbf{h}^i \in \mathbb{R}^n$ and $p_i \in \mathbb{R},\ i = 1, 2, \ldots, I$. $H^i \succeq 0$ means that H^i is defined as a positive semi-definite matrix. The positive semi-definiteness of H^i makes the constraint $g_i(\mathbf{x}) \leq 0$ a convex quadratic constraint. When $H^i = 0$, for some i, the constraint becomes a linear constraint ${\mathbf{h}^i}^T \mathbf{x} \leq -p_i$.

Our goal is to design an algorithm seeking the global solutions and maintaining the feasibility in each iteration. In this way, the algorithm always provides meaningful solutions even when it stops prematurely.

Before the algorithm starts, some parameters should be set up to make EM-CQ work correctly. The main parameters are listed as follows:

Parameters

Δ_{tol}: the stopping tolerance.

Δ: the stopping parameter.

$\phi > 1,\ 0 < \theta < 1$: the increasing and decreasing factors, respectively.

r: the initial population size.

$d_{\text{res}},\ N_{\text{res}}$: the thresholds for restarting the search.

Restart: the indicator for restarting the search, initially set as $false$.

Successful improvement: the indicator for global search.

After the parameters have been set, the algorithm goes to its major steps. The major structure of the EM algorithm for solving convex quadratically constrained problems of the form (6.1) is similar to the one for linearly constrained problems in Chapter 3. EM-CQ contains four major steps as well: *Initialize*, *Local*, *CalcF* and *Move*. Recall that in EM-Lin, there are four different methods for generating the initial feasible points. Three of them work for EM-CQ with some modifications and the details will be shown in the corresponding section. A modified local search method is implemented in *Local* step to achieve better performance. The core step of the EM method — *CalcF* — remains the same, while the original *Move* procedure may produce points violating the convex quadratic constraints. Therefore, some adjustments are needed. Moreover, a restart procedure is added to improve the efficiency of EM-CQ.

In the next three sections we will focus on *Initialization*, *Local Search* Method, *Move* procedure, *Restart Criterion and Termination Criteria*.

6.2 Initialization

Before starting the main iterates of the algorithm, a group of initial points in

$$\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n |\ g_i(\mathbf{x}) \leq 0,\ i = 1, 2, \ldots, I,\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\} \tag{6.3}$$

are needed.

The basic idea of generating initial feasible populations in EM-CQ is similar to the one in EM-Lin. The difference is that the feasible region is no longer a polyhedron. Hence

Algorithm 12 EM for Convex Quadratic Constrained Problem

Define Parameters
Initialize. Randomly generate the initial population $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^r\}$ in the convex feasible region $\mathcal{S}$.
Evaluate the initial active points $f(\mathbf{x}^i),\ i = 1, 2, \ldots, r$.
$\mathbf{x}^{\text{best}} = \mathbf{x}^1,\ f^{\text{best}} = f(\mathbf{x}^{\text{best}})$.
$Successful\ improvement = false$, and $Restart = false$.
while the stopping criteria are not satisfied **do**
  if $Restart = true$ **then**
    Regenerate the initial population, keep the current $\mathbf{x}^{\text{best}}$ and Δ.
    Set $Restart = false$.
  end if
  Set $\mathbf{x}_0^{\text{best}} = \text{argmin}\{f(\mathbf{x}^i),\ i = 1, 2, \ldots, r\}$.
  if $f(\mathbf{x}_0^{\text{best}}) < f^{\text{best}}$ **then**
    Set $f^{\text{best}} = f(\mathbf{x}_0^{\text{best}})$, $\mathbf{x}^{\text{best}} = \mathbf{x}_0^{\text{best}}$ and $Successful\ improvement = true$.
    $\Delta = \phi \times \Delta$ (increase the stopping parameter).
    Skip the **Local Search** step and go to **CalcF(ν)**.
  else
    Apply **Local Search** for convex quadratic constraints at $\mathbf{x}^{\text{best}}$ which provides updated $\mathbf{x}^{\text{best}}$, f^{best} and $Successful\ improvement$.
    if $Successful\ improvement = false$ **then**
      $\Delta = \theta \times \Delta$ (decrease the stopping parameter).
    end if
  end if
  CalcF(ν). Compute total force vectors by applying Algorithm 10.
  Move in Convex Feasible Region.
  Check **Restart Criterion** and **Stopping Criteria.** Update $Restart$.
  if $\Delta < \Delta_{\text{tol}}$ **then**
    STOP. Return $\mathbf{x}^{\text{best}}$ and f^{best}.
  end if
end while

the steps in EM-CQ should be modified.

The first strategy is to ignore the convex quadratic constraints and randomly generate points in the region shaped by the linear constraints and/or simple bounds (if no linear constraints). Then a newly generated point is accepted if it satisfies the quadratic constraints. This strategy is straightforward and easy to implement. However, such a strategy may not be efficient for generating a diverse feasible population. When the feasible region formed by the quadratic constraints is relatively small compared to the region formed by the linear constraints and/or simple bounds, the probability that a randomly generated point falls in the feasible region would be low. We keep this method of generating initial feasible population in our algorithm and use it when other methods fail to provide the initial feasible population.

There is another method for generating the initial feasible population. Since the constraint functions are all convex quadratic, we may produce several convex quadratic programming problems by using the feasible domain $\mathcal{S}$ and artificial linear functions as the objective functions. Then we apply the interior point method for linear conic programming problems to solve the problems. During the solving procedure, by recording the location of the point in each iteration, we are able to obtain some feasible solutions. Moreover, the convex combinations of these solutions can be used as the initial feasible points. Since the convex quadratic programming problems can be solved in polynomial time under the assumption that there exists a strict interior feasible point, this method may finish generating the initial feasible population in polynomial time.

The third way is explained below. First, find an interior point $\mathbf{x}^0$ that lies inside the

feasible domain $\mathcal{S}$ by solving a quadratic programming problem:

$$\begin{aligned} \max \quad & t \\ \text{s.t.} \quad & g_i(\mathbf{x}) + t \leq 0, \ i = 1, 2, \ldots, I, \end{aligned} \tag{6.4}$$

with an optimal solution $(\mathbf{x}^*, t^*)$. If $t^* > 0$, $\mathbf{x}^*$ can be accepted as $\mathbf{x}^0$. Then, from $\mathbf{x}^0$, a set of vectors $\{\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^r\}$ pointing to different directions are generated. If we choose the step length α_i carefully so that these vectors do not reach the boundary $\partial\mathcal{S}$, the trial points

$$\mathbf{x}^i = \mathbf{x}^0 + \alpha_i \mathbf{v}^i, \ \alpha_i > 0, \ i = 1, 2, \ldots, r, \tag{6.5}$$

can be treated as the initial feasible points. This method was used in some algorithms, for example, the Genetic Algorithm solver and Pattern Search solver in the MATLAB toolbox. Some modifications have been done in these solvers and the directions $\{\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^r\}$ are more diverse than the ones which are randomly generated.

Recall that in EM-Lin, there is a fourth method to generate the initial populations, that is, to find a maximum-volume ellipsoid which is inscribed in a polyhedron. Unfortunately, it could not be modified and adopted in EM-CQ. Since finding a maximum-volume ellipsoid that is inscribed in a convex set is generally not an easy task [67] and currently we have not found an efficient software to deal with this problem.

In the algorithm, we first apply the third method. If the calculated step lengths in the method are too small, which means the generated points are too close to $\mathbf{x}^0$ in (6.5), we turn to the second method. If the second method still does not provide enough initial feasible solutions, the first method is then utilized.

Algorithm 13 Initialize($\mathcal{S}, r$)

$\mathcal{S}$: the feasible region given by the problem.
r: the number of points in the population.

Set N_2 and N_3 as the thresholds for methods $2, 3$, respectively.
Apply method 3. Stop when the # of initial feasible solutions $= r$.
if the step length is smaller than N_3 **then**
  Apply method 2. Stop when the # of initial feasible solutions $= r$.
  if the # of recorded feasible solution is smaller than N_2 **then**
    Apply method 1. Stop when the # of initial feasible solutions $= r$.
    if the # of initial feasible solutions $< r$ **then**
      Return error.
    end if
  end if
end if
Output the initial feasible solutions $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^r\}$.

6.3 Local Search

The procedure *Local* is used to gather local information at a given point $\mathbf{x}$ and thus find better solutions in the neighborhood of $\mathbf{x}$. The local search method used here is based on the one used in EM-Lin that searches the local optimal solution around $\mathbf{x}^{\text{best}}$.

Before the iteration starts, some parameters should be set up.

Parameters

Δ: the stopping parameter for the outer algorithm.

α_s: the stopping threshold for local search.

$\alpha > 0$: the step length.

$\phi_\Delta,\ \phi_\alpha > 1$: the increasing factors.

$\rho_\alpha < 1$: the decreasing factor for step length.

$Maxiter$: the maximum number of iterations allowed in local search.

Local success: the indicator for the local search.

In each iteration of the local search method, we evaluate the objective function values

of points $\{\mathbf{x}_L^i,\ i = 1, 2, \ldots, k\}$ which lie in the neighborhood of $\mathbf{x}^{\text{best}}$. The new points $\{\mathbf{x}_L^i\}$ are obtained by adding a set of feasible directions $\{\mathbf{j}^i,\ i = 1, 2, \ldots, k\}$ to $\mathbf{x}^{\text{best}}$,

$$\mathbf{x}_L^i = \mathbf{x}^{\text{best}} + \alpha_i \times \mathbf{j}^i,\ i = 1, 2, \ldots, k, \tag{6.6}$$

where α_i is the step length.

Then among the set of points $\{\mathbf{x}^{\text{best}}$ and $\mathbf{x}^i,\ i = 1, 2, \ldots, k\}$, we pick the one with the lowest function value as the new $\mathbf{x}^{\text{best}}$. If the new $\mathbf{x}^{\text{best}}$ is different from the current one, which means that an improvement is made, the step length is increased to make a more aggressive search. Otherwise, when there is no better point found, the step length is decreased. When the step length is smaller than a threshold or the maximum number of iterations has been reached, the procedure stops and returns the point with the lowest objective value. The maximum number of local search steps grows as EM-CQ proceeds in order to perform more local searches as the current best point is closer to the global minimum. The details of the local search method is given here.

In order to make the algorithm simple and fast, the quadratic constraints are not considered in the calculation of $\{\mathbf{j}^i,\ i = 1, 2, \ldots, k\}$. Thus the calculation is the same as the one in EM-Lin (Algorithm 9). Note that the output point remains feasible since the extreme penalty function keeps the feasibility of every output point.

6.4 Calculation of Total Force Vector and Movement

The calculating procedure *CalcF* for EM-CQ is the same as the one used in the original EM and EM-Lin. For details, refer to Algorithm 4.

After calculating a normalized force vector $\mathbf{f}$ at a given feasible point $\mathbf{x}$, we can move

Algorithm 14 Local Search ($\mathbf{x}^{\text{best}}$, Δ), $Maxiter$

$\mathbf{x}^{\text{best}}$: the current best point.
Δ: the stopping parameter.
$Maxiter$: the maximum number of iterations in local search.

Let $F(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \in \mathcal{S}, \\ +\infty, & \text{otherwise.} \end{cases}$
iteration = 1, $F^{\text{best}} = F(\mathbf{x}^{\text{best}})$, *Local success* = $false$.
while $\alpha \geq \alpha_s$ or iteration $<$ $Maxiter$ **do**
  Compute the matrix J formed by the set of directions, $J = [\mathbf{j}^1, \ldots, \mathbf{j}^k]$, at $\mathbf{x}^{\text{best}}$.
  for $i = 1$ to k **do**
    if $F(\mathbf{x}^{\text{best}} + \alpha \times \mathbf{j}^i) < F^{\text{best}}$ **then**
      Set $\mathbf{x}^{\text{best}} = \mathbf{x}^{\text{best}} + \alpha \times \mathbf{j}^i$, $F^{\text{best}} = F(\mathbf{x}^{\text{best}} + \alpha \times \mathbf{j}^i)$.
      Set $\alpha = \alpha \times \phi_\alpha$ (increase the step length).
      Local success = *true*, break.
    end if
  end for
  if no better point than the current $\mathbf{x}^{\text{best}}$ for all $i = 1, 2, \ldots, k$ **then**
    Set $\alpha = \alpha \times \rho_\alpha$ (decrease the step length).
  end if
  iteration = iteration + 1.
end while
if *Local success* = *true* **then**
  Return the current $\mathbf{x}^{\text{best}}$.
  Set $\Delta = \Delta \times \phi_\Delta$ (increase the stopping parameter). Set *Successful improvement* = *true*.
else
  Stop the Local search method, set *Successful improvement* = $false$ and return the original $\mathbf{x}^{\text{best}}$.
end if
Increase $Maxiter$.

to a new position along the direction of $\mathbf{f}$:

$$\mathbf{x}_{\text{new}} = \mathbf{x} + \lambda RNG \times \mathbf{f},\ \lambda \sim U(0,1], \tag{6.7}$$

and our major task is to find the range parameter RNG.

In EM-CQ, the basic mechanism of movement of $\mathbf{x}$ is similar to that of EM-Lin. But instead of moving inside the feasible region formed by linear constraints, the points have to shift inside a region formed by quadratic constraints.

First, the calculating procedure of the range parameter in EM-Lin is kept in EM-CQ and the range parameter is recorded as RNG_L. The following step is to calculate the maximum step length for quadratic constraints.

Given the convex quadratic constraint functions $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_I(\mathbf{x}))$, we calculate the maximum step length for each component

$$g_i(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T H^i \mathbf{x} + \mathbf{h}^{iT}\mathbf{x} + p_i \tag{6.8}$$

to obtain its range, RNG_C^i, $i = 1, 2, \ldots, I$ as explained below. Then the maximum step length is chosen to be the smallest value of RNG_C^i and RNG_L, i.e.,

$$RNG = \min\{M,\ RNG_L,\ RNG_C^i,\ i = 1, 2, \ldots, I\}, \tag{6.9}$$

where M is a large positive number that prevents RNG from going to infinity. The following step is to compute the range for every $g_i(\mathbf{x}),\ i = 1, 2, \ldots, I$.

Given $\mathbf{x}$ and $\mathbf{f}$, for a constraint

$$g_i(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T H^i \mathbf{x} + \mathbf{h}^{iT}\mathbf{x} + p_i, \tag{6.10}$$

the range for $g_i(\mathbf{x})$, defined as RNG_C^i, is the largest value that satisfies the following inequality:

$$\frac{1}{2}(\mathbf{x}+RNG_C^i\mathbf{f})^T H^i(\mathbf{x}+RNG_C^i\mathbf{f})+\mathbf{h}^{iT}(\mathbf{x}+RNG_C^i\mathbf{f})+p_i\leq 0. \tag{6.11}$$

Equivalently,

$$\begin{aligned}
&(\frac{1}{2}\mathbf{f}^T H^i\mathbf{f})\times(RNG_C^i)^2+(\mathbf{f}^T H^i\mathbf{x}+\mathbf{h}^{iT}\mathbf{f})\times RNG_C^i+(\frac{1}{2}\mathbf{x}^T H^i\mathbf{x}+\mathbf{h}^{iT}\mathbf{x}+p_i)\\
\triangleq\ & a_{\text{move}}^i\times(RNG_C^i)^2+b_{\text{move}}^i\times RNG_C^i+c_{\text{move}}^i \qquad (6.12)\\
\leq\ & 0,
\end{aligned}$$

where a_{move}^i, b_{move}^i and c_{move}^i are known. It can be seen that $a_{\text{move}}^i\geq 0$, since H^i is a positive semi-definite matrix. Another observation is that $c_{\text{move}}^i=g_i(\mathbf{x})\leq 0$, because $\mathbf{x}$ is a feasible point. The quantity RNG_C^i can be calculated in the following way:

If $a_{\text{move}}^i=0$, then the inequality becomes $b_{\text{move}}^i\times RNG_C^i+c_{\text{move}}^i\leq 0$.

If $b_{\text{move}}^i\leq 0$, since $c_{\text{move}}^i\leq 0$, RNG_C^i can be any positive number.

If $b_{\text{move}}^i>0$, $RNG_C^i=-\dfrac{c_{\text{move}}^i}{b_{\text{move}}^i}$.

If $a_{\text{move}}^i>0$, it's a quadratic inequality for RNG_C^i. First calculate

$$\Delta_{\text{move}}^i=(b_{\text{move}}^i)^2-4a_{\text{move}}^i c_{\text{move}}^i. \tag{6.13}$$

Because $a_{\text{move}}^i\geq 0$ and $c_{\text{move}}^i\leq 0$, it can be seen that $4a_{\text{move}}^i c_{\text{move}}^i\leq 0$ and $\Delta_{\text{move}}^i\geq 0$. Thus

$$RNG_C^i=\frac{-b_{\text{move}}^i+\sqrt{\Delta_{\text{move}}^i}}{2a_{\text{move}}^i}. \tag{6.14}$$

Apply the above procedure for all quadratic constraints $g_i(\mathbf{x}) \leq 0,\ i = 1, 2, \ldots, I$, and through formula (6.9), we can find the step length at a corresponding point.

Algorithm 15 Move($\mathbf{x},\ \mathbf{f}$)

$\mathbf{x}$: the initial point.
$\mathbf{f}$: the force vector exerted on $\mathbf{x}$.
M: step length threshold.

Calculate RNG_L as in EM-Lin (Apply this even when no linear constraint exists, because bounded constraints are also considered in this step).
$RNG = \min\{M,\ RNG_L\}$.
for $i = 1$ to I **do**
  Set $a = \frac{1}{2}\mathbf{f}^T H^i \mathbf{f},\ b = \mathbf{f}^T H^i \mathbf{x} + \mathbf{h}^{iT}\mathbf{f}$ and $c = \frac{1}{2}\mathbf{x}^T H^i \mathbf{x} + \mathbf{h}^{iT}\mathbf{x} + p$.
  if $a = 0$ **then**
    if $b \leq 0$ **then**
      $RNG_C^i = M$.
    else
      $RNG_C^i = -\dfrac{c}{b}$.
    end if
  else
    $RNG_C^i = \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$.
  end if
  if $RNG_C^i < RNG$ **then**
    $RNG = RNG_C^i$.
  end if
end for
Output $\mathbf{x}_{\text{new}} = \mathbf{x} + \lambda RNG \times \mathbf{f},\ \lambda \in (0, 1]$.

6.5 Restart Criterion and Termination Criteria

As a meta-heuristic method, EM-CQ may not be effective if too many points stay close to each other. Especially, since the points are attracted by the current best point, after some iterations, they tend to move nearer and nearer to $\mathbf{x}^{\text{best}}$. To prevent this situation,

the algorithm restarts its search when many points are close to $\mathbf{x}^{\text{best}}$. If the Euclidian distance between a point $\mathbf{x}^i$ and $\mathbf{x}^{\text{best}}$ is smaller than a threshold d_{res}, it is considered to be too close to $\mathbf{x}^{\text{best}}$. When the number of close points exceeds a limit N_{res}, the algorithm generates new initial population and restarts the search while keeping the current $\mathbf{x}^{\text{best}}$ and Δ unchanged.

In EM-CQ, the algorithm stops when either the number of iterations or the number of function evaluations exceeds the corresponding limit, or when the stopping parameter Δ is less than a certain threshold. From Algorithm 12, it can been indicated that Δ will be decreasing when neither the search procedure nor the Local process is able to find a better point. Hence a sufficiently small Δ suggests that the algorithm could not improve and it is the time to terminate.

Algorithm 16 Check Restart Criterion (d_{res}, N_{res})

d_{res}: the threshold for testing whether $\mathbf{x}$ is close to $\mathbf{x}^{\text{best}}$.
N_{res}: the threshold for testing whether the algorithm should restart.

1: **for** $i = 1$ to r **do**
2: Calculate the distance between $\mathbf{x}^i$ and $\mathbf{x}^{\text{best}}$: $d_i = \|\mathbf{x}^i - \mathbf{x}^{\text{best}}\|$.
3: **end for**
4: **if** $\sum I_{(d_i < d_{\text{res}})} \geq N_{\text{res}}$, where $I_{(\cdot)}$ is the indicator function **then**
5: $Restart = true$.
6: **end if**

Chapter 7

Computational Experiments of EM-CQ

In this chapter, we compare EM-CQ (Algorithm 12) with other existing derivative free optimization solvers on convex quadratically constrained problems. In the first section, the test problems are introduced. They are either benchmark problems in the literature or randomly generated cases. Then, a comparison between EM-CQ and EM with different penalty functions is presented. Finally we present an investigation of the performances of EM-CQ and various derivative free optimizers, including the Genetic Algorithm, Particle Swarm Optimizer and Pattern Search Algorithm.

7.1 Selection and Generation of Test Problems

There are 40 test problems to be tested in this chapter. These problems can be divided into two groups. The first group of test problems consists of some common benchmark problems selected from Runarsson and Yao [92] and Zhang, Luo and Wang [114].

The second group of testing problems are the ones with the objective functions selected from CUTEr collection [37], Moshirvaziri [72] and Runarsson and Yao [92], while the constraints are randomly generated by several overlapped ellipsoids. The procedures of generating the constraints are described below.

First, a rectangle with a lower bound **lb** and an upper bound **ub** in the n-dimensional space is generated, where n is the dimension of the variables. Note that **lb** and **ub** are different from the lower bound **l** and upper bound **u** for the feasible region. Our goal is to generate a feasible region $\mathcal{S}$ that contains this rectangle so that $\mathcal{S}$ is not empty. A point c is randomly chosen and an ellipsoid is generated by

$$E_c(\mathbf{x}) = \left\{ \mathbf{x} \in \mathbb{R}^n \;\middle|\; \sum_{j=1}^{n} \frac{(x_j - c_j)^2}{\max\{ub_j - c_j,\ c_j - lb_j\}^2} \leq n \right\}. \tag{7.1}$$

Note that for a point $\mathbf{x}^0$ that is inside the rectangle, i.e., $x_j^0 \in [lb_j\ ub_j]$ for $j = 1, 2, \ldots, n$,

$$\text{if } x_j^0 \leq c_j, \text{ then } |x_j^0 - c_j| = c_j - x_j^0 \leq c_j - lb_j \leq \max\{ub_j - c_j,\ c_j - lb_j\},$$

$$\text{if } x_j^0 > c_j, \text{ then } |x_j^0 - c_j| = x_j^0 - c_j \leq ub_j - c_j \leq \max\{ub_j - c_j,\ c_j - lb_j\}.$$

Thus

$$\frac{(x_j - c_j)^2}{\max\{ub_j - c_j,\ c_j - lb_j\}^2} \leq 1,\ j = 1, 2, \ldots, n, \tag{7.2}$$

and

$$\mathbf{x}^0 \in E_c(\mathbf{x}). \tag{7.3}$$

Then the selected rectangle is covered by the ellipsoid. The constraint function is denoted

as

$$\begin{aligned} g_c(\mathbf{x}) &= \sum_{j=1}^{n} \frac{(x_j - c_j)^2}{\max\{ub_j - c_j,\ c_j - lb_j\}^2} - n \qquad (7.4) \\ &\triangleq \frac{1}{2}\mathbf{x}^T H_c \mathbf{x} + \mathbf{h}_c^T \mathbf{x} + p_c, \end{aligned}$$

where

$$H_c = \text{diag}\left\{\frac{2}{r_1^2},\ \frac{2}{r_2^2}, \ldots, \frac{2}{r_n^2}\right\},\ r_j = \max\{ub_j - c_j,\ c_j - lb_j\},\ j = 1, 2, \ldots, n, \qquad (7.5)$$

$$\mathbf{h}_c = -\left(\frac{2c_1}{r_1^2},\ \frac{2c_2}{r_2^2}, \ldots, \frac{2c_n}{r_n^2}\right)^T \text{ and} \qquad (7.6)$$

$$p_c = \sum_{j=1}^{n} \frac{c_j^2}{r_j^2} - n. \qquad (7.7)$$

The process is repeated until the desired number of constraints $\{g_i(\mathbf{x}),\ i = 1, 2, \ldots, I\}$, are generated. Since all the ellipsoids cover the original rectangle, the problem has a nonempty feasible region. Therefore, the general form of a test problem is written as follows:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \qquad (7.8) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0,\ i = 1, 2, \ldots, I, \end{aligned}$$

where $g_i(\mathbf{x}),\ i = 1, 2, \ldots, I$ are defined in (7.4) – (7.7).

There are in total 40 testing problems, including 7 benchmark problems (problems $1, 2, 6, 7, 8, 9$ and 10) and 33 randomly generated ones. The dimension of the problems ranges between 2 and 10 and the number of constraints ranges between 2 and 9.

7.2 Comparison of EM-CQ and EM Method with Different Penalty Functions

To solve a constrained optimization problem, one direct approach is using the penalty method. In this section, different types of penalty functions are utilized to make EM method applicable to the problems. Then we compare the performance of these penalty methods as well as the performance of EM-CQ on solving the test problems.

7.2.1 The Penalty Function Approaches

The penalty function method solves constrained optimization problems through a sequence of unconstrained optimization problems. The new penalized objective function $F_p(\mathbf{x})$ to be minimized is generally defined as

$$F_p(\mathbf{x}) = f(\mathbf{x}) + P(\mathbf{x}), \tag{7.9}$$

where $f(\mathbf{x})$ is the original objective function of the problem and $P(\mathbf{x})$ represents a penalty term. The function $P(\mathbf{x})$ is zero if no violation occurs, otherwise, it is positive.

Choosing an appropriate penalty function is a challenging work. If the penalty values are high, a minimization algorithm usually gets trapped by local minima. On the other hand, if the penalty value is low, it can hardly identify any feasible optimal solution. In the section, three major groups of penalty functions are proposed to handle the constraints in population-based heuristic methods of optimization: *death penalty approach* [8], *static penalty approach* [42, 55] and *adaptive penalty approach* [31, 38, 96, 112]. These methods are summarized as follows.

Death Penalty Approach The penalty function is defined in the following form:

$$P(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \text{ is feasible,} \\ +\infty, & \text{otherwise.} \end{cases} \tag{7.10}$$

The approach is simple and easy to implement. When the feasible region is reasonably large compared to the whole search space, this approach mostly works well [71]. However, when the feasible region is relatively small compared to the region formed by the lower and upper bounds, it spends a great amount of time searching for a feasible solution which makes it inefficient.

Static Penalty Approach The penalty parameters do not depend on the current iteration number and a constant penalty is applied to the infeasible solutions. Homaifar et al. [42] have proposed a static penalty approach in which users describe some levels of violation. A point $\mathbf{x}$ is evaluated by

$$F_p(\mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^{I} R_{ij} \max\{0, g_i(\mathbf{x})\}^2, \tag{7.11}$$

where R_{ij} indicates the penalty coefficient corresponding to the j^{th} constraint and i^{th} violation level. The disadvantage of this method is the large number of parameters that must be set. Michalewicz [69] shows that the quality of solutions is very sensitive to the values of these parameters. To run the algorithm without manually modifying the parameter during one set of experiments, we simplify the static penalty function by taking R_{ij} to be a constant d,

$$F_p(\mathbf{x}) = f(\mathbf{x}) + d \sum_{j=1}^{I} \max\{0, g_i(\mathbf{x})\}^2. \tag{7.12}$$

In our experiment, d is set to be 100 for all cases.

Adaptive Penalty Approach The penalty parameters are updated for every generation according to information gathered from the population. Hadj-Alouane and Bean [38] evaluated an individual by the following formula:

$$P(\mathbf{x}) = d(t)H_p(\mathbf{x}). \tag{7.13}$$

where $d(t)$ depends on the iteration t and $H_p(\mathbf{x})$ depends on the current point $\mathbf{x}$. The penalty parameter $d(t)$ is updated at every iteration t,

$$d(t+1) = \begin{cases} \beta_1 d(t), & \text{if case \#1 happens,} \\ \beta_2 d(t), & \text{if case \#2 happens,} \\ d(t), & \text{otherwise,} \end{cases} \tag{7.14}$$

where $0 < \beta_1 < 1$ and $\beta_2 > 1$.

Case #1 denotes the case that all of the best points in the last k iterations are feasible, and Case #2 denotes the case that they are all infeasible. In other words, if the best individuals of the last k iterations are all feasible, the penalty term $d(t+1)$ for iteration $t+1$ decreases. If they are all infeasible, the penalty term is increased. Otherwise, the penalty term does not change. The main problem of this approach is how to choose k, β_1 and β_2. Based on the experimental results and suggestions from other literatures, we set $k = 10,\ \beta_1 = 0.95$ and $\beta_2 = 1.1$. Moreover, we consider that $d(1)$ equals to the value of parameter d in the static penalty function, that is, $d(1) = 100$.

In (7.13), the term $H_p(\mathbf{x})$ whose value depends only on $\mathbf{x}$ is defined as follows,

$$H_p(\mathbf{x}) = \sum_{i=1}^{I} \theta(q_i(\mathbf{x})) q_i(\mathbf{x})^{r(q_i(\mathbf{x}))}, \tag{7.15}$$

where $q_i(\mathbf{x}) = \max\{0,\ g_i(\mathbf{x})\},\ i = 1, 2, \ldots, I.$ $\theta(q_i(\mathbf{x}))$ is the assignment function,

$$\theta(q_i(\mathbf{x})) = \begin{cases} d, & \text{if } q_i(\mathbf{x}) < 10^{-5}, \\ 10d, & \text{if } 10^{-5} \le q_i(\mathbf{x}) < 10^{-3}, \\ 100d, & \text{if } 10^{-3} \le q_i(\mathbf{x}) < 1, \\ 1000d, & \text{if } q_i(\mathbf{x}) \ge 1, \end{cases} \tag{7.16}$$

where d is set to be equal to the initial penalty parameter $d(1)$. And $\gamma(q_i(\mathbf{x}))$ is called the power of penalty function,

$$\gamma(q_i(\mathbf{x})) = \begin{cases} 1, & \text{if } q_i(\mathbf{x}) < 1, \\ 2, & \text{otherwise.} \end{cases} \tag{7.17}$$

7.2.2 Computational Results

The EM method with penalty functions and EM-CQ are tested using the 40 test problems collected in Section 7.1. We solve each problem 10 times by each solver and compare the best, average and worst achieved results. In average, it takes about 10 seconds to solve one problem for each solver. So it takes about 4 hours to finish all the tests in the experiment for EM-CQ and EM with penalty functions. Because the penalty methods may stop at an infeasible solution, we also compare the solvers' abilities of finding feasible solutions. This is done by considering the percentage of feasible solutions obtained in 10 runs for each testing problem. In our test, a constraint $g_i(\mathbf{x})$ is assumed to be violated if

and only if $g_i(\mathbf{x}) > 10^{-5}$ and $\mathbf{x}$ is said to be infeasible.

The initial population in the EM method with penalties are uniformly generated in the rectangle formed by the lower and upper bounds. Moreover, the local search method used in each penalty approaches is similar to the one used in EM-CQ, except that the search is applied on the penalty function $F_p(x)$.

In the experiment, the population size of EM-CQ and penalty EM methods is set to be 20. The maximum number of function evaluations is set to be 30,000. The running time is not recorded here since we are interested in the number of function evaluations. The algorithms are coded in MATLAB and run on an IBM laptop with Intel(R) Core Duo CPU @ 2.40GHz and 3.0GB of memory. The computational results related to each penalty approach are provided as follows.

Completion Percentage of Penalty EM Methods

In many real world problems, the solutions have little meaning when they are infeasible, thus it is important for a solver to find feasible solutions with a high probability. Table 7.1 shows the number of infeasible solutions obtained by each method.

From Table 7.1, we observed that,

1) There are only 4 problems whose feasible solutions are found by the three penalty EM solvers in all the 10 runs: *P.4, P.21, P.30* and *P.36*. Also there is one problem (*P.31*) for which no penalty EM method can generate any feasible solution.

2) The death penalty method has achieved 55% feasible solutions in all test runs, the static penalty method 73%, and the adaptive penalty method 74%.

3) There are 18 problems in which the death penalty method has found feasible solutions in all 10 runs, while there are 13 problems in which the death penalty method could not find any feasible solution.

4) There are 9 problems in which the static penalty method has found feasible solutions in all 10 runs, while there is only 1 problem in which the static penalty method could not find any feasible solution.

5) There are 15 problems in which the adaptive penalty method has found feasible solutions in all 10 runs, while there is also only 1 problem in which the adaptive penalty method could not find any feasible solution.

The results indicate that the death penalty EM method is the best among the three penalty methods in finding all feasible solutions in 10 runs, while it is the worst in missing all feasible solutions in 10 runs. Thus its solving ability is less stable than others. The other two methods are better choices if we want to find at least one feasible solution in the procedure. Between the two methods, the adaptive penalty method is better in obtaining more feasible solutions.

In Figure 7.1, the x-axis is the lower bound of the number of the feasible solutions obtained for each problem, and the y-axis is the number of the problems corresponding to x-axis. For example, the height of the first column (from left to right) in group one (i.e., $x = 1$) is 27, which means that there are 27 problems in which the death penalty method has found at least one solution in 10 runs. On the other side, from Table 7.1 we can see that there are 40 - 27 = 13 problems where the death penalty method does not find any feasible solution (*P.6*, *P.9 – P.12*, *P.15*, *P.16*, *P.18*, *P.19*, *P.31*, *P.34*, *P.35* and *P.39*). The second and the third columns in group one of Figure 7.1 represent the corresponding numbers of the static penalty method and adaptive penalty method, respectively. The heights of these two columns are both 39, which means that both of the two methods have 39 problems whose feasible solution can be found at least once in 10 runs.

From the above observation, we can see that the adaptive penalty EM method and static penalty EM method have similar performance and are both good if we want to find

Table 7.1: Problems with infeasible solutions obtained by penalty methods

Problem	Death	Static	Adaptive	Problem	Death	Static	Adaptive
P.1	1	0	0	P.21	0	0	0
P.2	1	1	1	P.22	0	2	3
P.3	0	4	1	P.23	0	6	2
P.4	0	0	0	P.24	0	2	6
P.5	2	3	7	P.25	0	3	1
P.6	10	0	0	P.26	0	6	8
P.7	5	4	6	P.27	0	5	7
P.8	7	3	2	P.28	8	7	5
P.9	10	2	0	P.29	0	2	4
P.10	10	0	0	P.30	0	0	0
P.11	10	3	4	P.31	10	10	10
P.12	10	4	3	P.32	0	8	3
P.13	0	4	9	P.33	0	1	1
P.14	0	4	2	P.34	10	3	3
P.15	10	0	0	P.35	10	2	6
P.16	10	0	0	P.36	0	0	0
P.17	9	0	0	P.37	9	1	0
P.18	10	1	0	P.38	0	3	6
P.19	10	1	0	P.39	10	2	0
P.20	0	6	3	P.40	8	5	1

at least one feasible solution when solving a problem. The death penalty EM method is better than the other two for obtaining all feasible solutions in a test problem. This is because that the first column at $x = 10$ is higher than the other two in group 10.

Solution Quality

In this section, the of solution qualities of the penalty EM methods and EM-CQ are compared. We keep the feasible solutions obtained by the penalty methods as useful data to compare with EM-CQ's results.

The general results are shown in Table 7.2, which gives the numbers of the best and worst solutions obtained by each EM method with different criteria. For example, in

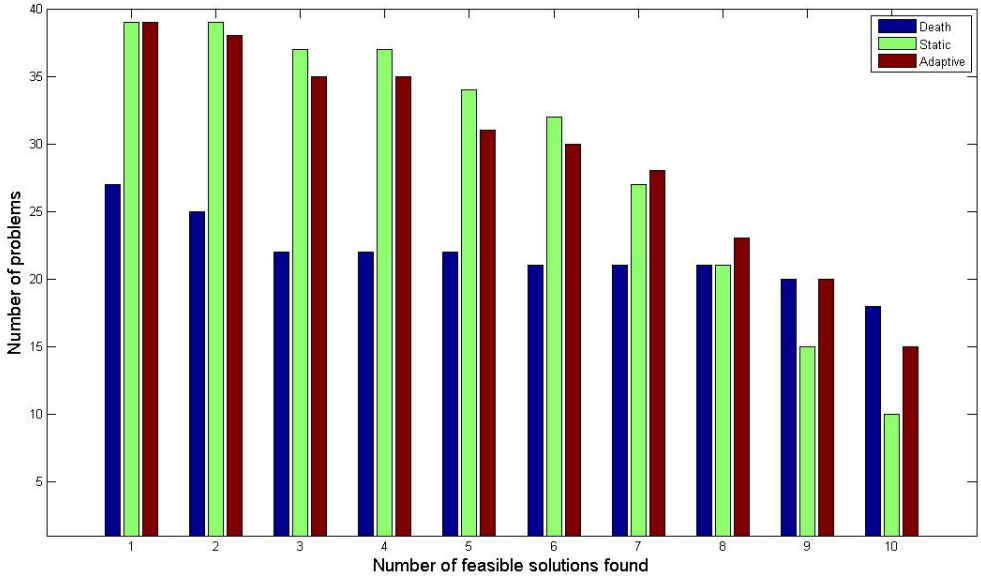

Figure 7.1: The ability of obtaining feasible solutions of different penalty methods

terms of the average objective function value, EM-CQ provides 33 best solutions among the 40 problems. At the same time, the adaptive penalty EM method provides 7 best solutions. In some problems more than one method have achieved the best solution so the total number of the best solutions is more than 40 in each row.

Table 7.2: Numbers of the best and worst solutions obtained by different EM methods

	the # of best solutions				the # of worst solutions			
Criterion	EM-CQ	Death	Static	Adaptive	EM-CQ	Death	Static	Adaptive
average	33	5	5	7	4	31	6	7
minimum	31	6	8	13	6	31	11	6
maximum	34	4	5	6	4	34	4	6

The results indicate that EM-CQ is the best method among these 4 methods in solving

the test problems. It produces most best solutions in terms of the average, minimum and maximum objective function values. Moreover, it has the fewest worst solutions under the same criteria. The performance of the adaptive penalty EM method and the static penalty EM method are similar. The adaptive method is slightly better than the static method. Especially, in terms of the minimum objective function value, the adaptive penalty EM method has the best solutions in 13 problems. At last, the death penalty EM method is the most unstable one among these 4 methods. It works fine in providing best solutions, but it is much worse in providing worst solutions. Part of the reason is that the death penalty EM method cannot give any feasible solution for some testing problems.

Table 7.3 shows the standard deviations of different EM methods. The bold number is the number with smallest standard deviation. In the calculation, if the number of feasible solutions achieved by a method is less than 2, its standard deviation for the corresponding problem is counted as not available and presented as N/A in the table. We find that the solutions of EM-CQ have the smallest standard deviations in 33 problems. The numbers for death, static and adaptive penalty EM methods are 7, 8 and 7, respectively. Hence EM-CQ is the most stable method among the 4 tested ones.

Table 7.3: Standard deviations for different EM methods

P	EM-CQ	Death	Static	Adaptive	P	EM-CQ	Death	Static	Adaptive
1	**0.0000**	**0.0000**	**0.0000**	**0.0000**	21	**0.0000**	**0.0000**	**0.0000**	**0.0000**
2	**0.0000**	103.5965	0.1437	0.2323	22	20.8231	**0.0000**	41.5615	30.4403
3	**0.0007**	0.0254	0.0329	0.0293	23	0.3018	40.3683	**0.0011**	0.4894
4	**0.0000**	**0.0000**	**0.0000**	**0.0000**	24	2.7423	**0.0000**	13.0755	5.5610
5	**0.0000**	0.3989	0.1618	0.0140	25	**12.3940**	74.4706	59.0836	34.8805
6	**3.2797**	N/A	25.7362	115.6010	26	**8.2606**	272.5063	170.8636	322.7960
7	**0.0000**	0.0341	0.0314	0.0298	27	**1.6548**	35.3595	13.6027	16.3494
8	**1.6926**	563.7942	17.1186	45.4657	28	**14.4149**	218.8965	155.3814	66.8714
9	0.0762	N/A	**0.0696**	0.1704	29	**9.1072**	161.8166	61.0703	62.2212
10	**0.0000**	N/A	0.0009	0.0017	30	**0.0000**	**0.0000**	**0.0000**	**0.0000**
11	**0.1948**	N/A	3.0078	3.3785	31	**0.0047**	N/A	N/A	N/A
12	**0.9197**	N/A	5.9842	8.0137	32	**2.1258**	15.5842	11.8183	6.9702
13	**0.1673**	1.6890	1.3913	N/A	33	0.0003	6642.5591	**0.0000**	**0.0000**
14	**1.2210**	15.7766	11.7021	4.5116	34	**95.4061**	N/A	258.3052	440.1025
15	**6.1334**	N/A	47.4558	35.2849	35	**49.6674**	N/A	568.6532	513.0258
16	**0.0412**	N/A	0.1165	0.2941	36	0.0362	0.0419	0.0483	**0.0361**
17	**0.7144**	N/A	1.0340	2.2843	37	**0.0000**	N/A	**0.0000**	**0.0000**
18	**0.3100**	N/A	3.2748	2.8879	38	**0.1476**	1.8253	0.8136	0.2781
19	**0.0294**	N/A	0.4137	0.6857	39	**0.0041**	N/A	0.0247	0.0469
20	3.0147	**0.0000**	5.7073	3.8932	40	**0.0570**	0.4874	0.1863	0.1795

Performance Profile Recall that in Section 4.2.1, given a solver $s \in \mathbb{S}$, a scalar $\tau \in [0, 1]$ and the set of test problems $\mathbb{P}$, the probability that the performance ratio $r_{p,s}$ is less than τ is defined as

$$\rho_s(\tau) = \frac{1}{n_p}(\text{the \# of elements in } \{p \in \mathbb{P} |\ r_{p,s} \leq \tau\}), \tag{7.18}$$

where n_p is the number of problems in the set $\mathbb{P}$ and the performance ratio for solution quality is

$$r_{p,s} = \frac{f_{p,s} - f_p^*}{f_p^w - f_p^*} (\leq 1), \tag{7.19}$$

where $f_{p,s}$ is the (average, minimum or maximum) objective function value of problem p obtained by solver s in 10 runs, f_p^* and f_p^w are the best and worst objective values for problem p among the solutions of all solvers, respectively. By definition, we can see that $r_{p,s}$ being small means that solver s provides a good solution with low objective function value for problem p. Especially, when $r_{p,s} = 0$, the solution given by s is the best solution for problem p. In (7.18), for a given τ, $\rho_s(\tau)$ is larger when the the # of elements in $\{p \in \mathbb{P} | r_{p,s} \leq \tau\}$ is larger.

Figures 7.2 — 7.5 show the performances of EM-CQ and EM method different penalty functions. From these figures it can be seen that EM-CQ is very competitive in terms of objective function evaluations.

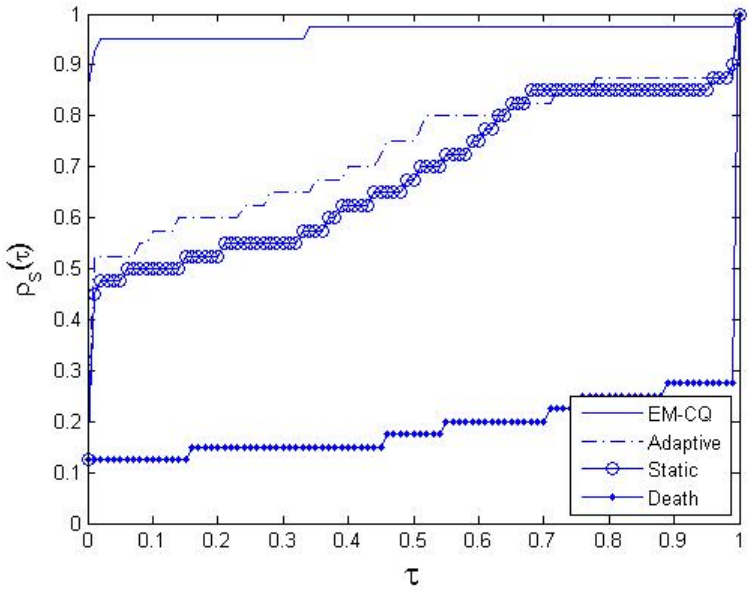

Figure 7.2: Performance profile of the average objective function values: EM-CQ vs. penalty EM methods

Figure 7.2 shows the performance profiles of different EM methods for the average objective function values. In this figure, we can see that the line of EM-CQ is always above the other three lines. It reaches 0.9 when τ is smaller than 0.1, which means that 90% of the solutions provided by EM-CQ are either the best solutions or quite close to the best solutions ($r_{p,\text{EM-CQ}} \leq 0.1$). Moreover, the line of EM-CQ starts at about 0.85, which shows that EM-CQ offers the best solutions in about 85% of the problems. As suggested previously, the adaptive penalty EM method is slightly better than the static penalty EM method since the line of the former is in most of the time higher than the later. Especially when $\tau \leq 0.6$, the difference is more apparent. And the death penalty EM method is the worst method in the average behavior.

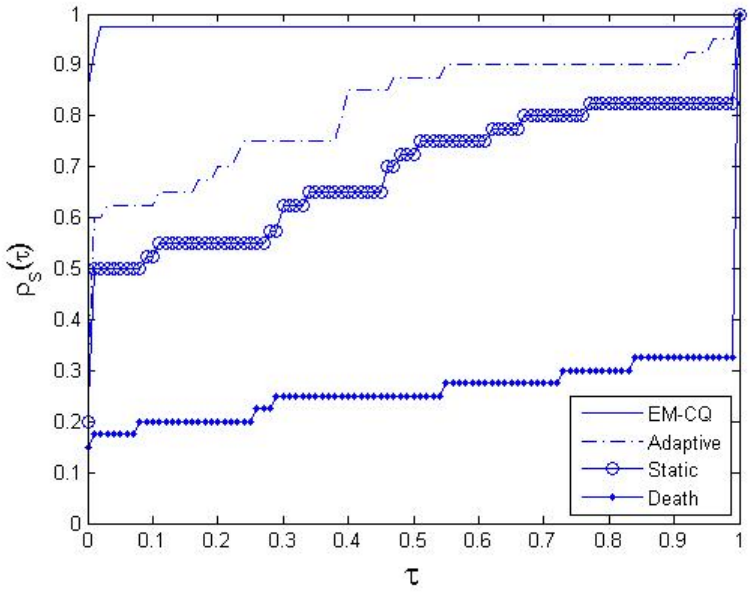

Figure 7.3: Performance profile of the minimum objective function values: EM-CQ vs. penalty EM methods

Figure 7.3 shows the performance profile of different EM methods for the minimum objective function values. The similar conclusion can be drawn as from Figure 7.2. EM-CQ still has the best performance and 95% solutions produced by EM-CQ are either the best ones or very close to them. Compared to Figure 7.2, the performance of adaptive penalty EM method is better, which means that this penalty method is also a good choice if one wants to simply find a best solution and does not care about other solutions. Also, the line of adaptive penalty EM method starts at 0.4 but jumps to 0.6 very fast. We can conclude that it provides the best answer in 40% of problems and gives answers that are very close to the best in 20% of problems.

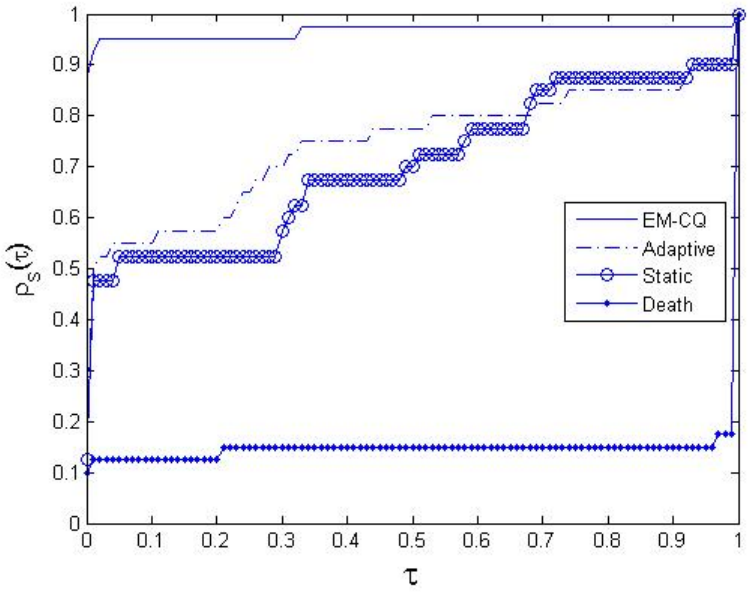

Figure 7.4: Performance profile of the maximum objective function values: EM-CQ vs. penalty EM methods

Figure 7.4 shows the performance profiles of different EM methods for the maximum objective function values. Not surprisingly, in this figure, the order of methods from the best to the worst is EM-CQ, adaptive penalty EM method, static penalty EM method and death penalty EM method.

Figure 7.5 shows the band of 95% confidence intervals of each solver. It can be seen that the band of EM-CQ is above all the other bands and do not overlap with them, which means that EM-CQ is significantly better than the EM method with different penalty functions.

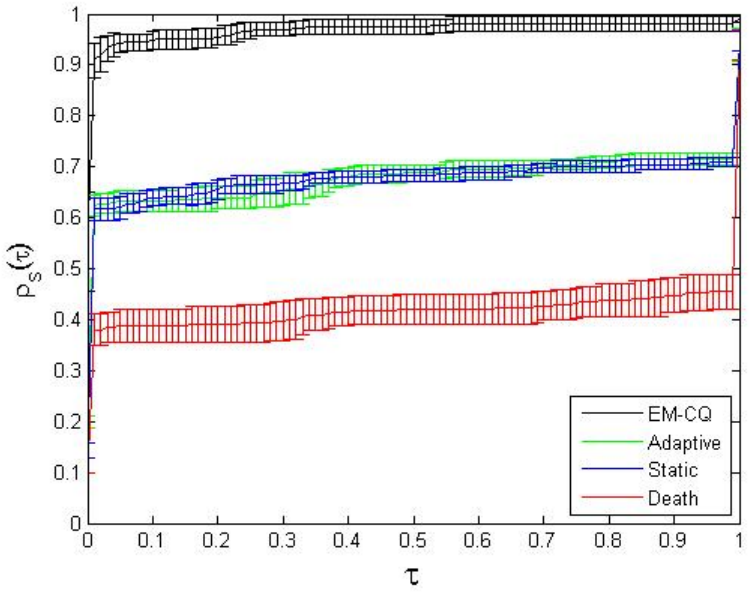

Figure 7.5: Bands of 95% confidence intervals for performance profile of the maximum objective function values: EM-CQ vs. penalty EM methods

7.2.3 Conclusion of EM-CQ vs. Penalty EM Methods

The results support the conclusion that EM-CQ is better than those penalty EM methods in solving convex quadratically constrained problems, while the adaptive penalty EM method is the best one among the three penalty EM methods. This conclusion is expected because EM-CQ takes the advantage of the special structure of convex quadratic constraints while the penalty methods do not. Since the constraints are quadratic, the step length in EM-CQ can be calculated explicitly, and the convex feasible region allows EM-CQ to move the points without violating feasibility.

7.3 Comparison of EM-CQ and Existing Global Optimizers

In this section, EM-CQ is compared with other heuristic methods, including the Genetic Algorithm (GA), Particle Swarm Optimizer (PSO) and Pattern Search Algorithm (PS). In the following sections, the three solvers, GA, PSO and PS, are introduced, modified and then run to solve the problems which have been tested by EM-CQ and penalty EM methods in Section 7.2. The results are used to compare with those obtained by EM-CQ.

The solvers used in our numerical experiments include Genetic Algorithm solver in the MATLAB toolbox [16, 17, 35], modified PSwarm (PSO) and Pattern Search Algorithm in the MATLAB toolbox [59]. The first two solvers have been introduced in Section 4.1. The Genetic Algorithm is tested without any change since it is designed to handle nonlinear constraints.

Some necessary modifications have been applied to the original PSwarm which is described in [109] to make it capable of solving convex quadratically constrained problems. The changes are on initialization part and movement part. These two processes are replaced by the ones which have been used in EM-CQ. In this manner all initial points are feasible and all points keep the feasibility as the number of iteration grows. Another adjustment for PSwarm is on the local search procedure, the local search method used in EM-CQ is adopted in PSwarm so that we can concentrate on the essential parts of the two algorithms.

Pattern Search Algorithm (PS) is a MATLAB toolbox for solving general constrained optimization problems. The algorithm is an adaption of a bounded constrained augmented Lagrangian method proposed by Conn, Gould and Toint in [16]. The algorithm solves the linearly constrained subproblem using a pattern search method and then relate the

stopping criterion with pattern size so that the convergence properties can be preserved.

In our experiment, the population size in EM-CQ, GA and PSO is set to be 20, because, according to our results, they have done best with this population size. The solvers are run 10 times for each problem, with the average, minimum and maximum solutions recorded for further analysis. In each run, the maximum number of function evaluations allowed is 30,000. The experiment is done on an IBM laptop with Intel(R) Core Duo CPU @ 2.40GHz and 3.0GB of memory.

7.3.1 Performances on Test Problems

The 7 selected test problems from [92, 114] are solved by the four solvers. Some of the problems have been modified to fit the requirement of being convex quadratically constrained. Tables 7.4, 7.5, 7.6 and 7.7 show the computational results of using EM-CQ, GA, PSO and PS, respectively. The problem numbers are the indices of the 40 tested problems. In the tables, n is the dimensionality of a problem, m_1 is the number of linear constraints and m_2 is the number of convex quadratic constraints.

Table 7.4: Computational results of using EM-CQ on solving problems from literature

P.	n	m_1	m_2	Known Best	Minimum	Average	95% Confidence Interval
1	2	0	1	0.0000	0.0000	0.0000	[0.0000, 0.0000]
2	2	0	1	16.5015	16.5015	16.5016	[16.5016, 16.5016]
6	10	3	5	24.3060	24.7657	27.7197	[25.8668, 29.5726]
7	2	0	2	-0.0958	-0.0958	-0.0958	[-0.0958, -0.0958]
8	7	0	4	460.6270	461.7287	463.4863	[462.5618, 464.4109]
9	9	0	9	-0.8660	-0.8657	-0.7220	[-0.7814, -0.6626]
10	3	0	1	-0.7383	-0.7383	-0.7383	[-0.7383, -0.7383]

Table 7.4 shows that for problems with small n and m, such as Problems 1, 2, 7

Table 7.5: Computational results of using GA on solving problems from literature

P.	n	m_1	m_2	Known Best	Minimum	Average	95% Confidence Interval
1	2	0	1	0.0000	0.0000	0.0000	[0.0000, 0.0000]
2	2	0	1	16.5015	16.5077	16.7210	[16.5461, 16.8959]
6	10	3	5	24.3060	27.8153	56.0996	[36.7798, 75.4193]
7	2	0	2	-0.0958	-0.0958	-0.0892	[-0.1035, -0.0748]
8	7	0	4	460.6270	463.5140	469.4439	[465.2102, 473.6776]
9	9	0	9	-0.8660	-0.8660	-0.7879	[-0.8576, -0.7182]
10	3	0	1	-0.7383	-0.7383	-0.7382	[-0.7383, -0.7382]

Table 7.6: Computational results of using PSO on solving problems from literature

P.	n	m_1	m_2	Known Best	Minimum	Average	95% Confidence Interval
1	2	0	1	0.0000	0.0000	0.0144	[-0.0093, 0.0382]
2	2	0	1	16.5015	16.5019	17.1639	[16.1591, 18.1687]
6	10	3	5	24.3060	25.3945	41.7825	[20.6579, 62.9071]
7	2	0	2	-0.0958	-0.0958	-0.0585	[-0.0854, -0.0315]
8	7	0	4	460.6270	484.9832	512.1282	[501.2280, 523.0284]
9	9	0	9	-0.8660	-0.7957	-0.7734	[-0.8212, -0.7256]
10	3	0	1	-0.7383	-0.7382	-0.7361	[-0.7382, -0.7340]

Table 7.7: Computational results of using PS on solving problems from literature

P.	n	m_1	m_2	Known Best	Minimum	Average	95% Confidence Interval
1	2	0	1	0.0000	0.0000	0.0000	[0.0000, 0.0000]
2	2	0	1	16.5015	16.5345	18.0846	[17.2094, 18.9598]
6	10	3	5	24.3060	25.1505	31.7719	[22.6315, 40.9123]
7	2	0	2	-0.0958	-0.0958	-0.0410	[-0.0673, -0.0147]
8	7	0	4	460.6270	471.6928	569.1471	[505.1495, 633.1446]
9	9	0	9	-0.8660	-0.8617	-0.8177	[-0.8461, -0.7893]
10	3	0	1	-0.7383	-0.7380	-0.7359	[-0.7379, -0.7339]

and 10, EM-CQ is able to find the known best solutions in most runs. When n and m increase, especially when m_2 increases, the problems become harder and EM-CQ encountered some difficulties in solving them. But still, the minimum answers produced by EM-CQ are close to the known bests.

From Tables 7.5 — 7.7, we can see that the accuracy of the solutions from GA, PSO

and PS is also highly related to the problem dimensions and number of constraints, especially the average and maximum solutions. The positive side is that most of the minimum solutions obtained are near the optimum. Combining Tables 7.4 — 7.7, we find that EM-CQ has competitive performances compared to the other three in terms of the objective function values.

Table 7.8: Objective function values: EM-Lin vs. other existing optimizers

P.	$f_a^{EM} - f_a^{GA}$ 95% Confidence Interval	$f_a^{EM} - f_a^{PSO}$ 95% Confidence Interval	$f_a^{EM} - f_a^{PS}$ 95% Confidence Interval
1	[0.0000, 0.0000]	[-0.0395, 0.0059]	[0.0000, 0.0000]
2	[-0.4038, -0.0351]	[-0.3459, 0.0213]	[-2.5055, -0.6605]
6	[-49.9212, -6.8384]	[-9.2898, -1.2356]	[-13.9859, 5.8815]
7	[-0.0218, 0.0084]	[-0.0658, -0.0089]	[-0.0826, -0.0271]
8	[-10.6781, -1.2371]	[-60.3440, -36.9398]	[-173.0615, -38.2600]
9	[-0.0378, 0.1697]	[-0.0196, 0.1225]	[0.0178, 0.1736]
10	[0.0000, 0.0000]	[-0.0044, 0.0001]	[-0.0045, -0.0003]

Table 7.8 shows the 95% confidence intervals of the differences between EM-CQ and other optimizers. Between EM-CQ and GA, there are 3 problems out of 7 where the 95% confidence intervals are to the left of 0, which means that in these 3 problems, EM-CQ is significantly better than GA. Similarly, EM-CQ is significantly better than PSO in 3 problems and EM-CQ is significantly better than PS in 4 problems.

According to the 95% confidence intervals of the differences between EM-CQ and other algorithms on all the 40 test problems, there are 23 problem in which EM-CQ is significantly better than GA, and 1 problem in which GA is significantly better than EM-CQ. There are 21 problem in which EM-CQ is significantly better than PSO, and 3 problems in which PSO is significantly better than EM-CQ. there are 28 problem in which EM-CQ is significantly better than PS, and 4 problems in which PS is significantly

better than EM-CQ.

Table 7.9: Numbers of the best solutions obtained by different heuristic methods

Criterion	EM-CQ	GA	PSO	PS
average	34	4	5	4
minimum	23	13	8	9
maximum	35	4	6	5

Table 7.10: Numbers of the worst solutions obtained by different heuristic methods

Criterion	EM-CQ	GA	PSO	PS
average	3	20	20	2
minimum	6	16	23	1
maximum	2	20	22	2

After examining the performance of EM-CQ, GA, PSO and PS of solving the selected problems from the literature, we continue the work for solving all test problems. The numbers of the best and worst solutions obtained by each heuristic method are shown in Tables 7.9 and 7.10.

From Table 7.9 we can see that EM-CQ performs well. In terms of the average and maximum objective values, more than 75% best solutions are achieved by EM-CQ. In terms of the minimum objective value, EM-CQ obtains about 60% of the best solutions. GA performs slightly better than PSO and PS in minimum solutions obtained. The three solvers function similarly in the average and maximum solutions obtained in 10 runs.

Table 7.10 indicates that PS does the best work in terms of the number of the worst solutions obtained. EM-CQ is also quite competitive.

Figures 7.6, 7.7 and 7.8 show the performance profiles of the average, minimum and

maximum objective function values found by the four tested solvers in 10 runs.

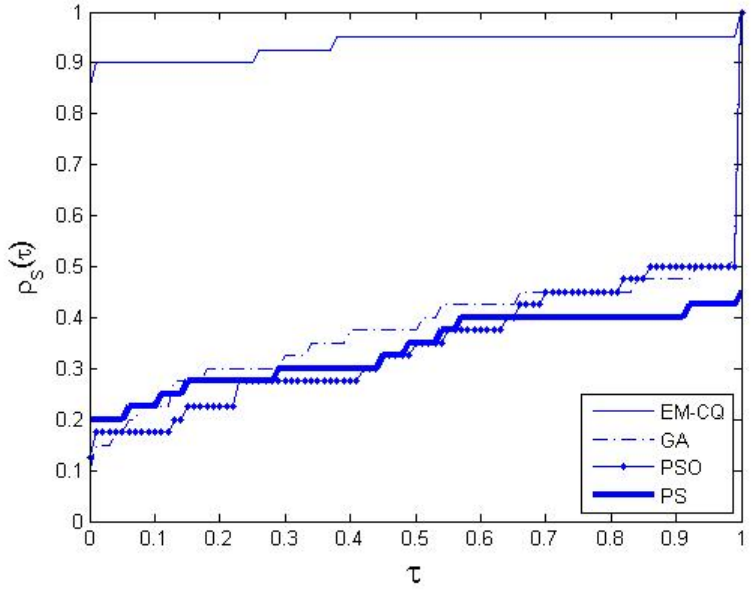

Figure 7.6: Performance profile of the average objective function values: EM-CQ vs. other methods

Figure 7.6 indicates that EM-CQ outperforms the other three solvers in terms of the average objective function values obtained. The line of EM-CQ reaches 0.9 almost instantly, which means that more than 90% of the average answers attained by EM-CQ are the best solutions. In other words, more than 90% of the answers obtained by EM-CQ have small $r_{p,EM-CQ} = \frac{f_{p,s}-f_p^*}{f_p^w-f_p^*}$.

In Figure 7.6, it can be seen that the lines of GA, PSO and PS are quite close. 30% of the average solutions obtained by GA have $r_{p,GA} \leq 0.2$, about 24% of the average solutions obtained by PSO have $r_{p,PSO} \leq 0.2$ and 28% of the average solutions obtained by PS have $r_{p,PS} \leq 0.2$. However, all of the three lines do not go above 0.5 before they

reach the end of the figure.

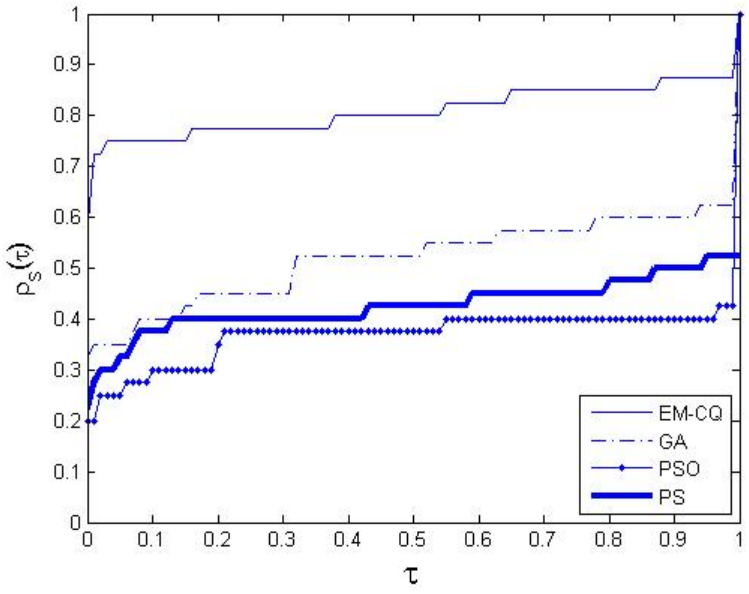

Figure 7.7: Performance profile of the minimum objective function values: EM-CQ vs. other methods

In Figure 7.7, the line of EM-CQ starts at a little bit higher than 60% and EM-CQ has 75% of solutions that have $r_{p,EM-CQ}$ smaller than 0.1. GA performs better than it does in the previous figure and the lines between EM-CQ's and GA's are closer. GA, PSO and PS, respectively, have about 40%, 30% and 38% of the solutions whose $r_{p,s}$ are smaller than 0.1. The figure shows that EM-CQ is still the best choice of obtaining a minimum solution in 10 runs while GA is not a bad one.

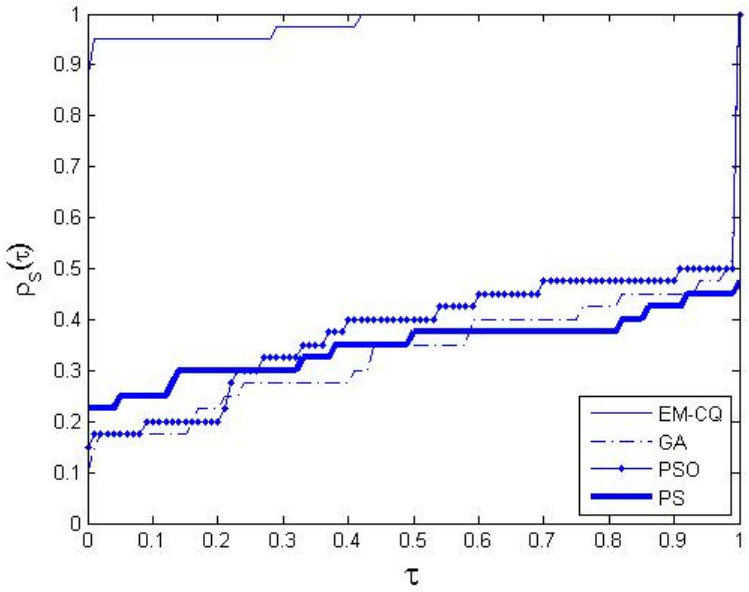

Figure 7.8: Performance profile of the maximum objective function values: EM-CQ vs. other methods

The performance of EM-CQ in Figure 7.8 is the best among Figures 7.6 — 7.8. The line of EM-CQ goes above 0.9 very fast and it goes to 1 when $\tau = 0.42$. This suggests that all the maximum solutions produced by EM-CQ have $r_{p,EM-CQ} \leq 0.42$. The other three solvers did not perform as well as EM-CQ for maximum solutions.

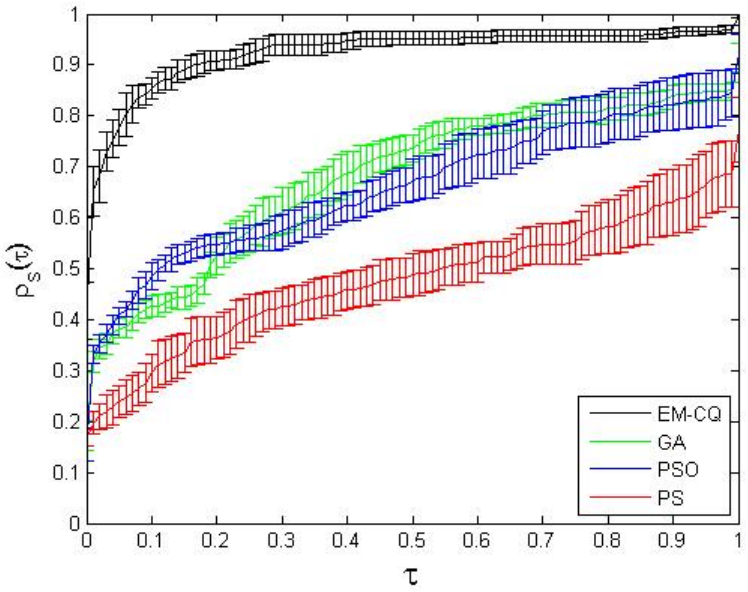

Figure 7.9: Bands of 95% confidence intervals for Performance profile of the objective function values: EM-CQ vs. other methods

Figure 7.9 shows the band of 95% confidence intervals for each solver. It can be seen that the band of EM-CQ is higher than others and do not overlap with them. It indicates that EM-CQ is significantly better than other existing solvers in solving the convex quadratically constrained problems. Moreover, the bands of GA and PSO overlap heavily, and both of them are higher than the band of PS.

Table 7.11 shows the standard deviations of the solutions found by EM-CQ, GA, PSO and PS. We find that the solutions of EM-CQ have the smallest standard deviations in 28 problems. The numbers for GA, PSO and PS are 6, 15 and 7, respectively. Hence EM-CQ is the most stable method of the four tested solvers.

Table 7.11: Standard deviations for different heuristic methods

P	EM-CQ	GA	PSO	PS	P	EM-CQ	GA	PSO	PS
1	**0.0000**	**0.0000**	0.0332	**0.0000**	21	**0.0000**	**0.0000**	**0.0000**	**0.0000**
2	**0.0000**	0.2445	1.4046	1.2234	22	**20.8231**	45.3280	105.8160	38.0201
3	**0.0007**	0.0047	0.7451	2.8027	23	**0.3018**	11.3130	13.6823	25.8872
4	**0.0000**	**0.0000**	**0.0000**	**0.0000**	24	2.7423	16.5097	**0.0000**	60.8732
5	**0.0000**	**0.0000**	**0.0000**	0.2689	25	**12.3940**	61.6698	58.5940	34.0826
6	**3.2797**	27.0071	26.6540	12.7774	26	**8.2606**	41.5480	83.0945	32.6444
7	**0.0000**	0.0200	0.0348	0.0368	27	**1.6548**	3.2689	1.9216	66.9563
8	**1.6926**	5.9183	34.4710	89.4624	28	**14.4149**	32.9383	184.1051	234.1512
9	0.0762	0.0974	**0.0248**	0.0396	29	**9.1072**	17.5088	43.2304	124.3739
10	**0.0000**	**0.0000**	0.0030	0.0028	30	**0.0000**	0.5567	**0.0000**	**0.0000**
11	0.1948	2.1943	**0.0000**	4.7862	31	0.0047	0.7378	**0.0000**	1.2213
12	**0.9197**	2.3575	3.2060	4.6757	32	2.1258	296.0000	**0.0553**	7.9694
13	**0.1673**	0.3234	0.5860	1.6570	33	0.0003	0.0917	**0.0000**	7068.5041
14	**1.2210**	5.1937	6.0172	12.6257	34	**95.4061**	668.9072	127.4402	298.3554
15	**6.1334**	249.8864	50.9126	137.6744	35	49.6674	314.2175	**10.0594**	211.4524
16	0.0412	0.1841	**0.0000**	0.2004	36	0.0362	0.0360	0.0364	**0.0000**
17	0.7144	1.9705	**0.0000**	0.9673	37	**0.0000**	**0.0000**	**0.0000**	**0.0000**
18	**0.3100**	2.5340	1.1544	5.3023	38	0.1476	0.0150	**0.0087**	0.5390
19	**0.0294**	0.1254	0.0550	0.0916	39	**0.0041**	0.0097	0.0060	0.0060
20	**3.0147**	4.7993	4.7376	7.1621	40	0.0570	0.5008	0.1205	**0.0483**

7.3.2 Conclusion of EM-CQ vs. Existing Global Optimizers

The experimental results suggest that EM-CQ performs better than the Genetic Algorithm toolbox in MATLAB, Particle Swarm Optimizer and Pattern Search Algorithm, for the 40 test problems. This means that the EM method is a very competitive heuristic and its structure is quite flexible to accommodate other searching procedures. Our modifications on the Particle Swarm Optimizer may not be a good fit to show its advantages. Moreover, PS is a single point heuristic whose performance depends on the starting point heavily.

7.4 Conclusion

In the first half of this chapter, we have studied several penalty methods for EM and compared them with EM-CQ by solving our convex quadratically constrained test problems. The experimental results show that the adaptive penalty EM method is the best penalty method for EM while EM-CQ outperforms all the selected penalty methods. In the second half of this chapter, we have compared EM-CQ with various heuristic approaches including Genetic Algorithm, Particle Swarm Optimizer and Pattern Search Algorithm. The analysis of the computational results shows that EM-CQ is a stable and competitive method. At the end, we gave some possible reasons for EM-CQ to perform better than GA, PSO and PS in the experiments.

Chapter 8

Conclusion and Further Research

In this chapter, we summarize the work we have done and point out some directions for future research.

8.1 Conclusion

We have reviewed a novel population-based, derivative free global optimization method — the Electromagnetism-like Mechanism (EM) method. The method imitates the behavior of electrically charged particles. The strength of the method lies in the idea of directing sample points toward local optimizers, which point out attractive regions of the feasible space.

The original EM method has been shown by numerical experience to have very competitive performance in solving global optimization problems using function evaluations only. However, it can only solve problems with box constraints, i.e., the problems with variables that have lower and upper bounds. This dissertation is directed to extend the EM method for solving problems with more complicated constraints, more specifically,

the linear and convex quadratic constraints.

The goal of this dissertation has been achieved in two steps. Firstly, we have developed the EM method for solving linearly constrained problems (EM-Lin). We have modified the *Initialize* and *Move* procedures as well as the *Terminating Criteria* to make the new algorithm capable of handling linear constraints. The *Local Search* part has also been adjusted to improve its efficiency. The experimental results have suggested that using function evaluations only, EM-Lin converges rapidly (in terms of the number of function evaluations) to the global optimum and produces highly efficient results for problems of varying degree of difficulty.

A long-term financial planning problem has been solved by implementing EM-Lin and the results are compared with the methods proposed in the literature. The results have indicated that EM-Lin is able to solve this highly nonconvex optimization problem.

In the second part of this dissertation, after analyzing the properties of quadratic functions, we have derived the EM method for solving convex quadratically constrained problems (EM-CQ). The major steps in EM-CQ have been redesigned to handle the constraints. Particularly, in the *Move* procedure, we have taken advantage of the properties of quadratic functions and derived a method for calculating the maximum step length in the feasible region. In this manner the feasibility can be maintained while the algorithm proceeds. We have also studied several penalty methods and implemented them with the original EM method to deal with nonlinear constraints and to compare with EM-CQ.

Our experimental results have shown that the adaptive penalty method for EM is the best one among the selected penalty methods for EM, and EM-CQ outperforms all the penalty methods for EM in solving the selected convex quadratically constrained problems. Moreover, our numerical experiments and comparisons between EM-CQ and other existing heuristic methods have indicated that EM-CQ is a competitive method for

providing effective results for convex quadratically constrained problems with different levels of difficulty.

8.2 Further Research

A future research direction is to handle general nonlinear constraints for the EM method. The structure of EM method is very flexible permitting the development of new variations. However, our experience has shown that the derivative free global optimization with general nonlinear constraints is very difficult to develop.

The first step could be attempting to solve general convex constrained problems since they are natural extensions of convex quadratically constrained problems. The step length may be calculated in the way similar to the one used in EM-CQ or using linear search method. The *Move* procedure will then follow and the algorithm can proceed iteratively.

When the constraints become nonconvex, the augmented Lagrangian method is a possible approach to overcome the difficulty. This method, which can be considered as an infeasible point method, is originally derived as a derivative based method [16, 15]. However, the idea can be adopted for derivative free optimization [59, 60]. The major idea is that, in each iteration of the method, the process consists of solving a sequence of subproblems. For each subproblem the augmented Lagrangian is approximately minimized in a domain defined by the linear constraints. Since EM-Lin has been developed, the minimization of the subproblems can be done using function evaluations only. And the whole problem is solved when a certain sufficient criticality is reached. While the specific definition of sufficient criticality needs to be further studied, the outline of the augmented Lagrangian method is listed below.

The global convergent property that has been proved for the augmented Lagrangian approach on derivative based algorithms. However, the global convergent property of augmented Lagrangian approach on EM method needs to be further developed.

Algorithm 17 Augmented Lagrangian Method

1: **Initialize.** Generate the initial solution. Estimate the Lagrangian multipliers and penalty parameter.
2: **Proceed Inner Iteration.** In each inner iteration, solve the linearly constrained subproblem.
3: **Test for Convergence.**
4: **Update Lagrangian multipliers and penalty parameter.**
5: **Begin a new outer iteration.**

Since the results in this dissertation have shown that the essential scheme utilized in EM method is quite efficient comparing to other heuristics such as GA, PSO and PS, good performance can be expected if the difficulty of handling nonlinear constraints can be overcome.

REFERENCES

[1] Al-Khayyal, F.A. and J.E. Falk (1983) Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8, 273-286.

[2] Anderberg, M.R. (1973) *Cluster Analysis for Applications*. Academic Press, New York, NY.

[3] Anstreicher, K.M. (2003) Recent advances in the solution of quadratic assignment problems. *Mathematical Programming*, 97(1-2), 27-42.

[4] Armijo, L. (1966) Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16, 1-3.

[5] Baba, N. (1981) Convergence of a random optimization method for constrained optimization problems. *Journal of Optimization Theory and Applications*, 33(4), 451-461.

[6] Baba, N., T. Shoman and Y. Sawaragi (1977) A modified convergence theorem for a random optimization method. *Information Sciences*, 13, 159-166.

[7] Bashir, Z.A. and M.E. El-Hawary (2009) Applying wavelets to short-term load forecasting using PSO-based neural networks. *IEEE Transactions on Power Systems*, 24(1), 20-27.

[8] Bäk, T., F. Hoffmeister and H.P. Schwell (1991) A Survey of evolution strategies. *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, 2-9.

[9] Benson, H.P. (1995) Concave minimization: theory, applications and algorithms. In R. Horst and P.M. Pardalos, editors, *Handbook of Global Optimization, Volume 2 of Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[10] Beveridge, R.S.G. (1970) *Optimization: Theory and Practice*. McGraw-Hill, New York, NY.

[11] Birbil, Ş.İ. (2002) Stochastic global optimization techniques. *PhD Thesis*, North Carolina State University, Raleigh, NC.

[12] Birbil, Ş.İ. and S.-C. Fang (2002) An electromagnetism-like mechanism for global optimization. *Journal of Global Optimization*, 25(3), 263-282.

[13] Birbil, Ş.İ., S.-C. Fang, and R.L. Sheu (2004) On the convergence of a population based global optimization algorithm. *Journal of Global Optimization*, 30(2), 301-318.

[14] Bonnans, J.F., J.C. Gilbert, C. Lemaréchal and C.A. Sagastizábal (2006) *Numerical Optimization: Theoretical and Practical Aspects.* Springer-Verlag, Berlin, Germany.

[15] Conn, A.R., N.I.M. Gould, A. Sartenaer and Ph.L. Toint (1996) Convergence properties of an augmented Lagrangian algorithm for optimization with a combination of general equality and linear constraints. *SIAM Journal on Optimization*, 6(3), 674-703.

[16] Conn, A.R., N.I.M. Gould and Ph.L. Toint (1991) A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2), 545-572.

[17] Conn, A.R., N.I.M. Gould and Ph.L. Toint (1997) A globally convergent augmented Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. *Mathematics of Computation*, 66(217), 261-288.

[18] Cowan, E.W. (1968) *Basic Electromagnetism.* Academic Press, New York, NY.

[19] Debels, D., B. De Reyck, R. Leus and M. Vanhoucke (2006) A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research*, 169, 638-653.

[20] Dekkers, A. and E. Aarts (1989) Global optimization and simulated annealing. *Mathematical Programming*, 50, 367-393.

[21] Dennis, J.E. Jr. and V. Torczon (1991) Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4), 448-474.

[22] Devroye, L. (1978) Progressive global random search of continuous functions. *Mathematical Programming*, 15, 330-342.

[23] Dill, K.A., A.T. Phillips and J.B. Rosen (1997) Protein structure prediction and potential energy landscape analysis using continuous global minimization. *Proceedings of the First Annual International Conference on Computational Molecular Biology*, 109-117.

[24] DIRECT – A Global Optimization Algorithm. Dan Finkel's Matlab implementation of DIRECT. `http://www4.ncsu.edu/~ctk/Finkel_Direct/`.

[25] Dolan, E.D. and J.J. Moré (2002) Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91, 201-213.

[26] Eberhart, R.C. and J. Kennedy (1995) A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, Nagoya, Japan, 39-43.

[27] Fletcher, R. and C. Reeves (1964) Function minimization by conjugate directions. *Computer Journal*, 7, 149-154.

[28] Floudas, C.A. (1999) *Deterministic Global Optimization: Theory, Methods and Applications, 2nd Edition, Volume 37 of Nonconvex Optimization and Applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[29] Floudas, C.A. and P.M. Pardalos (1990) A collection of test problems for constrained global optimization algorithms. *Lecture Notes in Computer Science 455*, Springer, Berlin, Germany.

[30] Fogel, L.J., A.J. Owens and M.J. Walsh (1966) *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, NY.

[31] Gen, M. and R. Cheng (1996) A survey of penalty techniques in genetic algorithms. *Proceedings of the 1996 International Conference on Evolutionary Computation, IEEE*, 804-809.

[32] Gill, P. and W. Murray (1972) Quasi-Newton methods for unconstrained optimization. *IMA Journal of Applied Mathematics*, 9(1), 91-108.

[33] GLOBAL Library, `http://www.gamsworld.org/global/globallib.htm`.

[34] Glover, F. and M. Laguna (1997) *Tabu Search*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[35] Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley Longman Publishing Company, Inc., Boston, MA.

[36] Goldfarb, D. and A. Idnani (1983) A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27, 1-33.

[37] Gould, N.I.M., D. Orban and Ph.L. Toint CUTEr, a constrained and unconstrained test environment, revisited. `http://cuter.rl.ac.uk/cuter-www`.

[38] Hadj-Alouane, A.B. and J.C. Bean (1997) A genetic algorithm for the multiple-choice integer program. *Operations Research*, 45, 92-101.

[39] Hansen, E.R. (1992) *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, NY.

[40] Hartigan, J. (1975) *Clustering Algorithms*. John Wiley, New York, NY.

[41] Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI.

[42] Homaifar, A., S.H.Y. Lai and X. Qi (1994) Constrained optimization via genetic algorithms. *Simulation*, 62, 242-254.

[43] Hooke, R. and T.A. Jeeves (1961) Direct search solution of numerical and statistical problems. *Journal of the ACM*, 8, 212-229.

[44] Horst, R. and H. Tuy (1993) *Global Optimization: Deterministic Approaches*. Springer-Verlag, Berlin, Germany.

[45] Huyer, W. and A. Neumaier (1999) Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14, 331-355.

[46] Jain, A.K. and R.C. Dubes (1988) *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ.

[47] Jardine, N. and R. Sibson (1971) *Mathematical Taxonomy*. Wiley, London, UK.

[48] Ji, Y., K.-C. Zhang and S.-J. Qu (2007) A deterministic global optimization algorithm. *Applied Mathematics and Computation*, 185, 382-387.

[49] Jones, D.R., C.D. Perttunen and B.E. Stuckman (1993) Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79, 157-181.

[50] Karmarkar, N. (1984) A new polynomial time algorithm for linear programming. *Combinatorica*, 4(4), 373-395.

[51] Kearfott, R.B. (1996) *Rigorous Global Search: Continuous Problems*. Kluwer Academic Publishers, Dordrecht / Boston / London.

[52] Kirkpatrick, A., A.K.C.J. Gelatt and M.P. Vechi (1983) Optimization by simulated annealing. *Science*, 220, 671-680.

[53] Kolda, T.G., R.M. Lewis and V. Torczon (2003) Optimization by direct search: new perspectives on some classical and modern methods. *SIAM Review*, 45, 385-482.

[54] Kostreva, M.M. Method of Feasible Directions (MFD). `http://www.math.clemson.edu/~flstgla/mfd.html`.

[55] Kuri-Morales, A. and C.C. Quezada (1998) A universal eclectic genetic algorithm for constrained optimization. *Proceedings 6th European Congress on Intelligent Techniques & Soft Computing, EUFIT'98*, 518-522.

[56] Land, A.H. and A.G. Doig (1960) An automatic method of solving discrete programming problems. *Econometrica*, 28, 497-520.

[57] Lee, C.-K. (2006) Global optimization of hybrid systems. *PhD Thesis*, Massachusetts Institute of Technology, Boston, MA.

[58] Lemke, C. (1962) A method of solution for quadratic programs, *Management Science*, 8, 442-453.

[59] Lewis, R.M. and V. Torczon (2002) A globally convergent augmented Lagrangian pattern search algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Optimization*, 12(4), 1075-1089.

[60] Lewis, R.M. and V. Torczon (2010) A direct search approach to nonlinear programming problems using an augmented Lagrangian method with explicit treatment of the linear constraints. *Submitted to SIAM Journal on Optimization.*

[61] Lucidi, S., M. Sciandrone and P. Tseng (1999) Objective-derivative-free methods for constrained optimization. *Mathematical Programming*, 92, 37-59.

[62] Luenberger, D.G. and Y. Ye (2008) *Linear and Nonlinear Programming, 3rd edition.* Springer-Verlag, New York, NY.

[63] Maranas, C.D., I.P. Androulakis, C.A. Floudas, J.M. Mulvey and A.J. Berger (1997) Solving long-term financial planning problems via global optimization. *Journal of Economics Dynamics and Control*, 21(8/9), 1405-1425.

[64] Maranas, C.D. and C.A. Floudas (1994) Global minimum potential energy conformations of small molecules. *Journal of Global Optimization*, 4, 135-170.

[65] Marwala, T. (2005) Finite element model updating using particle swarm optimization. *International Journal of Engineering Simulation*, 6(2), 25-30.

[66] Mehrotra, S. (1992) On the implementation of a primal-dual interior point method. *SIAM Journal of Optimization*, 2, 575-601.

[67] Mehrotra, S. and J. Sun (1991) On computing the center of a convex quadratically constrained set. *Mathematical Programming*, 50(1), 81-89.

[68] Michalewicz, Z. (1994) Evolutionary computation techniques for nonlinear programming problems. *International Transactions in Operational Research*, 1, 223-240.

[69] Michalewicz, Z. (1995) Genetic algorithms, numerical optimization, and constraints. *Proceedings of the 6th International Conference on Genetic Algorithms*, Pittsburgh, PA, 151-158.

[70] Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs, 3rd edition.* Springer, Berlin, Germany.

[71] Michalewicz, Z., D. Dasgupta, R.G. Le Riche and M. Schoenauer (1996) Evolutionary algorithms for constrained engineering problems. *Evolutionary Computation*, 4, 1-32.

[72] Moshirvaziri, K. (1998) Construction of test problems for concave minimization under linear and nonlinear constraints. *Journal of Optimization Theory and Applications*, 98, 83-108.

[73] Murty, K.G. (1988) *Linear Complementarity, Linear and Nonlinear Programming*. Helderman-Verlag, Berlin, Germany.

[74] Murty, K.G. and S.N. Kabadi (1987) Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39, 117-129.

[75] Nelder, J.A. and R. Mead (1965) A simplex method for function minimization, *The Computer Journal*, 7, 308-313.

[76] Neumaier, A. (1990) *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, UK.

[77] Neumaier, A. (2006) Global optimization and constraint satisfaction. In I. Bomze, I. Emiris, A. Neumaier and L. Wolsey, editors, *Proceedings of GICOLAG Workshop (of the Research Project Global Optimization, Integrating Convexity, Optimization, Logic Programming and Computational Algebraic Geometry)*.

[78] Neumaier, A. and J-P. Merlet (2002) Constraint satisfaction and global optimization in robotics. `http://www.mat.univie.ac.at/~neum/ms/robslides.pdf`.

[79] Nocedal, J. and S.J. Wright (2006) *Numerical Optimization, 2nd edition*. Springer-Verlag, New York, NY.

[80] Osman, I.H. and J.P. Kelly, editors (1996) *Meta-Heuristics: Theory and Applications.* Kluwer Academic Publishers, Dordrecht / Boston / London.

[81] Pardalos, P.M. and H.E. Romeijn, editors (2002) *Handbook of Global Optimization, Volume 2 of Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[82] Pauling, L. (1960) *The Nature of the Chemical Bond*. Cornell University Press, Ithaca, NY.

[83] Pintér, J.D. (1984) Convergence properties of stochastic optimization procedures. *Optimization*, 15(3), 405-427.

[84] Pinťer, J.D. (1996) *Global Optimization in Action: Continuous and Lipschitz Optimization–Algorithms, Implementations, and Applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[85] Ratschek, H. and J. Rokne (1988) *New Computer Methods for Global Optimization*. Ellis Horwood, Chichester, UK.

[86] Rechenberg, I. (1973) *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, Germany.

[87] Rinnooy Kan, A.H.G. and G.T. Timmer (1987) Stochastic global optimization methods part I: clustering methods. *Mathematical Programming*, 39, 27-56.

[88] Rinnooy Kan, A.H.G. and G.T. Timmer (1987) Stochastic global optimization methods part II: multi level methods. *Mathematical Programming*, 39, 57-78.

[89] Rinnooy Kan, A.H.G. and G.T. Timmer (1994) Stochastic methods for global optimization. *American Journal of Mathematical Management Science*, 4, 7-40.

[90] Rosen, J.B. (1960) The gradient projection method for nonlinear programming. part I. linear constraints. *Journal of the Society for Industrial and Applied Mathematics*, 8, 181-217.

[91] Rosen, J.B. (1961) The gradient projection method for nonlinear programming. part II. nonlinear constraints. *Journal of the Society for Industrial and Applied Mathematics*, 9, 514-532.

[92] Runarsson, T.P. and X. Yao (2000) Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4, 284-294.

[93] Schaffer, J.D., R.A. Caruana, L.J. Eshelman and R. Das (1989) A study of control parameters affecting online performance of genetic algorithms for function optimization. *Proceedings of the 3rd International Conference on Genetic Algorithms*, Fairfax, VA, 51-60.

[94] Schoen, F. (2002) Two-phase methods for global optimization. In P.M. Pardalos and H.E. Romeijn, editors, *Handbook of Global Optimization, Volume 2*, Kluwer Academic Publishers, Netherlands, 151-177.

[95] Schwefel, H.P. (1981) *Numerical Optimization of Computer Models*. Wiley and Sons Inc., Chichester, UK.

[96] Smith, A. and D. Tate (1993) Genetic optimization using a penalty function. *Proceedings of the 5th International Conference on Genetic Algorithms*, Urbana-Champaign, IL, 499-503.

[97] Sneath, P.H.A. and R.R. Sokal (1973) *Numerical Taxonomy.* Freeman, San Francisco, CA.

[98] Snyman, J.A. (2005) *Practical Mathematical Optimization.* Springer, New York, NY.

[99] Solis, F.J. and R.J-B. Wets (1981) Minimization by random search techniques. *Mathematics of Operations Research*, 6, 19-30.

[100] Strongin, R.G. and Y.D. Sergeyev (2000) *Global Optimization with Non-Convex Constraints: Sequential and Parallel Algorithms.* Kluwer Academic Publishers, Dordrecht / Boston / London.

[101] Taylor, A.E. and W.R. Mann (1972) *Advanced Calculus, 2nd edition.* John Wiley and Sons Inc., New York, NY.

[102] Thoai, N.V. (1988) A modified version of Tuys method for solving D.C. programming problems. *Optimization*, 19, 665-674.

[103] Törn, A., M.M. Ali and S. Viitanen (1999) Stochastic global optimization: problem classes and solution techniques. *Journal of Global Optimization*, 14, 437-447.

[104] Tuy, H. (1987) Global minimization of the difference of two convex functions. *Mathematical Programming Study*, 30, 150-182.

[105] Van de Panne, C. and A. Whinston (1964) The simplex and the dual method for quadratic programruing. *Operations Research Quarterly*, 15, 355-389.

[106] Vanderbei, R.J. Benchmarks for nonlinear optimization. `http://www.princeton.edu/~rvdb/bench.html`.

[107] Van Laarhoven, P.J.M. and E.H.L. Aarts (1987) *Simulated Annealing: Theory and Applications.* Kluwer Academic Publishers, Dordrecht, The Netherlands.

[108] Vavasis, S.A. (1991) *Nonlinear Optimization, Complexity Issues.* Oxford University Press, New York, NY.

[109] Vaz, A.I.F. and L.N. Vicente (2009) PSwarm: a hybrid solver for linearly constrained global derivative-free optimization. *Optimization Methods and Software*, 24(4-5), 669-685.

[110] Wolsey, L.A. and G.L. Nemhauser (1988) *Integer and Combinatorial Optimization.* John Wiley and Sons Inc., New York, NY.

[111] Wright, M.H. (2005) The interior-point revolution in optimization: history, recent developments, and lasting consequences. *Bulletin of the American Mathematical Society*, 42, 39-56.

[112] Yokota, T., M. Gen, K. Ida and T. Taguchi (1996) Optimal design of system reliability by an improved genetic algorithm. *Electronics and Communications in Japan, Part III: Fundamental Electronic Science*, 79(2), 41-51.

[113] Zhang, Y. and L. Gao (2001) On numerical solution of the maximum volume ellipsoid problem. *SIAM Journal on Optimization*, 14, 53-76.

[114] Zhang, M., W. Luo and X. Wang (2008) Differential evolution with dynamic stochastic selection for constrained optimization. *Information Sciences*, 178, 3043-3074.

[115] Zoutendijk, G. (1960) *Methods of Feasible Directions*. Elsevier Publishing Company, Amsterdam, The Netherlands.

APPENDICES

Appendix A

Sources of the benchmark test problems with linear constraints

avgasa: CUTEr collection with classification QLR2-AN-8-10.

avgasb: CUTEr collection with classification QLR2-AN-8-10.

biggsc4: CUTEr collection with classification QLR2-AN-4-7.

bunnag1, bunnag2, bunnag3, bunnag4, bunnag5, bunnag6, bunnag7, bunnag8, bunnag9, bunnag10, bunnag11, bunnag12 and **bunnag13**: problems 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 and 14 from Bunnag, D. and M. Sun (2005) Genetic algorithm for constrained global optimization in continuous variables. *Applied Mathematics and Computation*, 171, 604-636.

ex2_1_1, ex2_1_2, ex2_1_3, ex2_1_4, ex2_1_7 and **ex2_1_10**: problems from Floudas, C.A., P.M. Pardalos, C.S. Adjiman, W.R. Esposito, Z.H. Gumus, S.T. Harding, J.L. Klepeis, C.A. Meyer and C.A. Schweiger (1999) *Handbook of Test Problems in Local and Global Optimization.* Kluwer Academic Publishers Dordrecht, The Netherlands.

genocop07, genocop09, genocop10 and **genocop11**: problems 7, 9, 10 and 11 from

Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs, 3rd edition*. Springer, Berlin, Germany.

goffin: CUTEr collection with classification LLR2-AN-51-50.

hatfldh: CUTEr collection with classification QLR2-AN-4-7.

himmelbi: CUTEr collection with classification OLR2-MN-100-12.

hs021: CUTEr collection with classification QLR2-AN-2-1.

hs024: CUTEr collection with classification OLR2-AN-2-3.

hs035: CUTEr collection with classification QLR2-AN-3-1.

hs036: CUTEr collection with classification OLR2-AN-3-1.

hs037: CUTEr collection with classification OLR2-AN-3-2.

hs044: CUTEr collection with classification QLR2-AN-4-6.

hs076: CUTEr collection with classification QLR2-AN-4-3.

hs0118: CUTEr collection with classification QLR2-AN-15-17.

hs021mod: CUTEr collection with classification SLR2-AN-7-1.

hs035mod: CUTEr collection with classification QLR2-AN-3-1.

hs044new: CUTEr collection with classification QLR2-AN-4-6.

Ji1, Ji2 and **Ji3**: problems 1, 2 and 3 from Ji, Y., K.-C. Zhang and S.-J. Qu (2007) A deterministic global optimization algorithm. *Applied Mathematics and Computation*, 185, 382-387.

lsqfit: CUTEr collection with classification SLR2-AN-2-1.

makela4: CUTEr collection with classification LLR2-AN-21-40.

Michalewicz1: problem 3 from Michalewicz, Z. (1994) Evolutionary computation techniques for nonlinear programming problems. *International Transactions in Operational Research*, 1, 223-240.

nuffield_continuum: CUTE set and GAMS World collection.

pentagon: CUTEr collection with classification OLR2-AY-6-15.

pt: CUTEr collection with classification LLR2-AN-2-V.

simpllpa: CUTEr collection with classification LLR2-AN-2-2.

simpllpa: CUTEr collection with classification LLR2-AN-2-3.

sipow1: CUTEr collection with classification LLR2-AN-2-V.

sipow1m: CUTEr collection with classification LLR2-AN-2-V.

sipow2: CUTEr collection with classification LLR2-AN-2-V.

sipow2m: CUTEr collection with classification LLR2-AN-2-V.

tfi2: CUTEr collection with classification LLR2-AN-3-V.

weapons: modified problem from Bracken, J. and G.P. McCormick (1968) *In Selected Applications of Nonlinear Programming.* John Wiley and Sons Inc., New York, NY, 22-27. Reformulation has removed 1 variable (*objvar*) and 1 equation (*e13*).

zecevic2: CUTEr collection with classification QLR2-AN-2-2.

Appendix B

Modified benchmark problems with convex quadratic constraints

The problem indices are consistent with the indices in the experimental tests. P.1, P.2, P.6, P.7, P.8, P.9 and P.10 are the modified benchmark problems. Rest of the problems which are not listed here are the ones with randomly generated feasible regions.

P.1

$$\begin{aligned} \min \quad & f(\mathbf{x}) = (1 - x_1)^2 \\ \text{s.t.} \quad & x_1^2 - x_2 \leq 0 \\ & -20 \leq x_1, x_2 \leq 20. \end{aligned}$$

The optimal solution is $\mathbf{x}^* = (1, 1)$ with $f(\mathbf{x}^*) = 0$.

P.2

$$\begin{aligned} \min \quad & f(\mathbf{x}) = (x_1 - 5)^2 + x_2^2 \\ \text{s.t.} \quad & x_1^2 - x_2 \leq 0 \\ & -20 \leq x_1, x_2 \leq 20. \end{aligned}$$

The optimal solution is $\mathbf{x}^* = (1.2348, 1.5247)$ with $f(\mathbf{x}^*) = 16.5015$.

P.6

$$\begin{aligned} \min \quad & f(\mathbf{x}) = x_1^2 + x_2^2 + x_1 x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 \\ & \quad +2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \\ \text{s.t.} \quad & -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0 \\ & 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0 \\ & -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0 \\ & 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0 \\ & 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0 \\ & x_1^2 + 2(x_2 - 2)^2 - 2x_1 x_2 + 14x_5 - 6x_6 \leq 0 \\ & 0.5(x_1 - 8)^2 + 2(x_2 - 2)^2 + 3x_5^2 - 6x_6 - 30 \leq 0 \\ & -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0 \\ & -10 \leq x_i \leq 10, \ i = 1, 2, \ldots, 10. \end{aligned}$$

The optimal solution is $\mathbf{x}^* = (2.1720, 2.3637, 8.7739, 5.0960, 0.9907, 1.4306, 1.3216, 9.8287, 8.2801, 8.3759)$ with $f(\mathbf{x}^*) = 24.3062$.

P.7

$$\begin{aligned} \min \quad & f(\mathbf{x}) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1+x_2)} \\ \text{s.t.} \quad & x_1^2 - x_2 + 1 \leq 0 \\ & 1 - x_2 + (x_2-4)^2 \leq 0 \\ & 0 \leq x_1, x_2 \leq 10. \end{aligned}$$

The optimal solution is $\mathbf{x}^* = (1.2280, 4.2454)$ with $f(\mathbf{x}^*) = 0.0958$.

P.8

$$\begin{aligned} \min \quad & f(\mathbf{x}) = (x_1-10)^2 + 5(x_2-12)^2 + x_3^4 + 3(x_4-11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 \\ & \qquad -4x_6x_7 - 10x_6 - 8x_7 \\ \text{s.t.} \quad & -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0 \\ & -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0 \\ & -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0 \\ & 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0 \\ & -10 \leq x_i \leq 10, \ i = 1, 2, \ldots, 7. \end{aligned}$$

The optimal solution is $\mathbf{x}^* = (2.3305, 1.9514, -0.4775, 4.3657, -0.6245, 1.0381, 1.5942)$ with $f(\mathbf{x}^*) = 680.6301$.

P.9

$$
\begin{aligned}
\min \quad & f(\mathbf{x}) = -0.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7) \\
\text{s.t.} \quad & x_3^2 + x_4^2 - 1 \leq 0 \\
& x_9^2 - 1 \leq 0 \\
& x_5^2 + x_6^2 - 1 \leq 0 \\
& x_1^2 + (x_2 - x_9)^2 - 1 \leq 0 \\
& (x_1 - x_5)^2 + (x_2 - x_4)^2 - 1 \leq 0 \\
& (x_1 - x_7)^2 + (x_2 - x_8)^2 - 1 \leq 0 \\
& (x_3 - x_5)^2 + (x_4 - x_6)^2 - 1 \leq 0 \\
& (x_3 - x_7)^2 + (x_4 - x_8)^2 - 1 \leq 0 \\
& x_7^2 + (x_8 - x_9)^2 - 1 \leq 0 \\
& -10 \leq x_i \leq 10, \ i = 1, 2, \ldots, 8 \text{ and } 0 \leq x_9 \leq 20.
\end{aligned}
$$

The best solution known is $\mathbf{x}^* = (-0.6578, -0.1534, 0.3234, -0.9463, -0.6578, -0.7532, 0.3234, -0.3465, 0.5998)$ with $f(\mathbf{x}^*) = -0.8660$.

P.10

$$
\begin{aligned}
\min \quad & f(\mathbf{x}) = -\frac{100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2}{100} \\
\text{s.t.} \quad & (x_1 - 1)^2 + (x_2 - 2)^2 + (x_3 - 3)^2 - 0.0625 \leq 0 \\
& 0 \leq x_1, x_2, x_3 \leq 10.
\end{aligned}
$$

The optimal solution is $\mathbf{x}^* = (1.2000, 2.15003.1000)$ with $f(\mathbf{x}^*) = -0.7383$.