Nonsmooth and Derivative-Free Optimization Based Hybrid Methods and Applications

Qiang Long

This thesis is submitted in total fulfilment of the requirement for the degree of Doctor of Philosophy

School of Science, Information Technology and Engineering
Faculty of Science
Federation University Australia
PO Box 663
University Drive, Mount Helen
Ballarat, VIC 3353, Australia.

March 2014

Abstract

Nonsmooth and global optimization are among the most difficult subjects in optimization. Nonsmooth optimization problems appear in practical applications where the objective functions to be minimized/maximized or the constraint functions are not necessarily differentiable. There also exists the so-called "stiff problems" which are smooth analytically but nonsmooth numerically. Because gradients of the objective and constraint functions are discontinuous in nonsmooth optimization, smooth optimization methods, which require gradient information of the objective function and constraints, cannot be applied to solve such problems. Various methods have been developed to solve nonsmooth optimization problems, such as the subgradient and bundle-type methods. These methods are efficient for solving nonsmooth convex problems; however, they are not always efficient for solving nonsmooth nonconvex problems because they are based upon local convex models of functions.

In this thesis, we develop hybrid methods for solving global and in particular, nonsmooth optimization problems. Hybrid methods are becoming more popular in global optimization since they allow to apply powerful smooth optimization techniques to solve global optimization problems. Such methods are able to efficiently solve global optimization problems with large number of variables. To date global search algorithms have been mainly applied to improve global search properties of the local search methods (including smooth optimization algorithms). In this thesis we apply rather different strategy to design hybrid methods. We use local search algorithms to improve the efficiency of global search methods. The thesis consists of two parts. In the first part we describe hybrid algorithms and in the second part we consider their various applications.

Our first hybrid method is based on the quasisecant method of nonsmooth optimization. The quasisecant method is similar to the bundle-type methods; however, instead of using subgradients, the quasisecant method uses quasisecants to approximate subdifferential. The hybrid algorithm combines the quasisecant method and a special procedure for identifying "promising" basins of the objective function in the search space. This procedure is designed using the quasisecant method. It is proved that this algorithm converges to the global optimal solution if the number of local minimizers of the objective function is finite and they are isolated. Numerical results are presented to demonstrate the efficiency of the proposed algorithm.

Our second hybrid method is based on the combination of the genetic algorithm and the derivative-free Hooke Jeeves method. More specifically, an acceleration operator designed using the Hooke Jeeves method is embedded into the general procedure of the genetic algorithm. This acceleration operator improves the convergent rate and accuracy of the genetic algorithm. This method is applied to solve constrained global optimization problems.

In the second part of the thesis we apply the quasisecant method and the first hybrid method to solve some problems from applications. First, we develop new algorithm for solving the system of nonsmooth equations by modifying the quasisecant method. The system of nonsmooth equations is transformed into a nonsmooth optimization problem with a zero minimal objective function value. The quasisecant method is then applied to solve the nonsmooth optimization problem. Two different nonsmooth optimization models are studied and the convergent properties of the algorithm is presented. The numerical performance of the proposed algorithm is included.

Finally, the proposed hybrid method is applied to solve the molecular conformation and sensor localization problems. Numerical results show that this method is efficient and robust for such problems.

Statement of Authorship

This thesis contains no work extracted in whole or in part from a thesis, dissertation or research paper previously presented for another degree or diploma except where explicit reference is made. No other person's work has been relied upon or used without due acknowledgment in the main text and bibliography of the thesis.

Signed: [illegible]

Date: 21/03/2014

Acknowledgement

First, I would like to express my great thanks to my principal supervisor Prof. Adil Bagirov. The appreciation I have toward him is beyond words. It is a big honour for me to be his student and work with him during my candidature. I can never achieve this point without his constant patience and encouragement. What I have learned from him is far more than research, but how to be a good person and how to respect different ideas. I sincerely hope that I will still have a chance to work with him in my future academic career. Also, I would like to appreciate the efforts of my associate supervisor Dr. Alex Kruger. I have learned a lot from him at the early stage of my PhD study.

I would also like to give my sincere appreciation to Prof. Zhiyou Wu and Dr. Fusheng Bai. It was Prof. Zhiyou Wu who introduced me to Prof. Adil Bagirov and encouraged me to grip the chance of doing PhD at the University of Ballarat. Thanks for all the help and guidance from Prof. Zhiyou Wu and Dr. Fusheng Bai over the past four years of my study, which made my life so wonderful in Ballarat. I feel so lucky that I have some excellent mentors like them in my life. In addition, I would like to express my gratitude to all the colleagues and friends I have made in Ballarat; it was them who shared so much colorful time with me.

My project would not have been completed without the financial support of the School of SITE. I would like to express my thanks to the Dean of SITE, Prof. John Yearwood, the staff of SITE and Research Services for their extraordinary support.

Finally, my heartfelt appreciation goes to my families for their loving support and understanding.

Dedication

To my wife Jing Wen and daughter Emma

List of publication

Journal papers

1. Adil M. Bagirov, Qiang Long. A Heuristic Method for Global Optimization with Special Procedure for Exploring Basins. (Submitted)

2. Qiang Long, Junjian Huang. A New Hybrid Method Combining Genetic Algorithm and Coordinate Search Method. *2012 IEEE the fifth international conference on advanced computational intelligence (ICACI)*. October 18-20, Nanjing, Jiangsu, China, PP.1072-1077.

3. Qiang Long. The Application of Genetic Algorithm in Solving Nonsmooth Optimization Problems, *Journal of Chongqing Normal University (Natural Science Edition)*, 2013 (01), 12-15.

4. Qiang Long, Jueyou Li. Numerical Performance of Subgradient Methods in Solving Nonsmooth Optimization Problems. *Journal of Chongqiang Normal University (Natural Science Edition)*, 2013 (06), 25-30.

5. Qiang Long, Changzhi Wu. A Quasisecant Method for Solving a System of Nonsmooth Equations. *Computers and Mathematics with Applications*, Volume 66, 2013, 419-431.

6. Qiang Long, Changzhi Wu. A Hybrid Method Combining Genetic Algorithm and Hooke-Jeeves Method for Constrained Global Optimization. *Journal of Industrial and Management Optimization*, Volume 10, Number 4, October 2014, 1279-1296

7. Changzhi Wu, Chaojie Li, Qiang Long. A DC Programming Approach for Sensor Network Localization with Uncertainties in Anchor Positions. *Journal of Industrial and Management Optimization*, Volume 10, Number 3, July 2014, 817-826.

8. Qiang Long. A Constraint Handling Technique for Constrained Multi-objective Genetic Algorithm. *Swarm and Evolutionary Computation*. Volume 15, 2014, 66-79

9. Jueyou Li, Changzhi Wu, Zhiyou Wu, Qiang Long. Gradient-free Method for Nonsmooth Distributed Optimization. *Journal of Global Optimization*. DOI: 10.1007/s10898-014-0174-2. (Accepted)

10. Jueyou Li, Zhiyou Wu, Qiang Long. An Objective Penalty Function Approach for Solving Constrained Minimax Problems. *Journal of the Operations Research Society of China*. DOI: 10.1007/s40305-014-0041-3. (Accepted)

11. Changzhi Wu, Qiang Long. Distributed Mirror Descent Algorithm and Applications in Distributed Estimation. (Submitted)

12. Qiang Long, Changzhi Wu. A System of Nonsmooth Equations Solver Based upon Subgradient Method. (Submitted)

Conference and workshop presentations

1. 01/2011 -*Nonsmooth Numerical Shape Optimization*, the 2011 Winter School of Australian Mathematical Sciences Institute (AMSI) held in the University of Queensland.

2. 01/2012 -*Numerical Methods in Nonsmooth Optimization*, Chongqing Normal University, Chongqing, China.

3. 11/2012 -*An Improved Genetic Algorithm for Global Optimization*, the Annual Conference of Federation University Australia.

4. 07/2013 -*A Hybrid Method for Global Optimization*, the 26th European Conference on Operational Research.

5. 11/2013 -*A Constraint Handling Technique for Constrained Multi-objective Optimization*, the Annual Conference of Federation University Australia.

Contents

List of Tables

List of Figures

Introduction

Nonsmooth optimization is one of the most attractive subjects in optimization. In nonsmooth optimization problems, objective and/or constraint functions are assumed to be (locally Lipcshitz) continuous while their gradients can be discontinuous. The theory of nonsmooth optimization is based upon nonsmooth analysis which generalizes the definitions of gradient and other related concepts from smooth analysis. The necessary and sufficient optimality conditions for nonsmooth optimization problems are studied in nonsmooth analysis.

Over the last five decades, various deterministic methods have been developed for solving nonsmooth optimization problems. In most of these methods the sequence of approximate solutions $\{x_k\}$ is constructed as follows:

$$x_{k+1} = x_k - \alpha_k H_k g_k,$$

where x_k is the current iteration point, α_k is the step length, H_k is a symmetric matrix and g_k is the gradient or subgradient of the objective function at x_k. Let $d_k = -H_k g_k$, then d_k is called a search direction. There are two main phases in most algorithms for solving nonsmooth optimization problems: (i) finding the search direction and (ii) calculating the step length along the search direction.

Among all the deterministic methods for solving nonsmooth optimization problems, the subgradient method and bundle method (and its variations) are the most popular ones. The subgradient method is a simple extension of the gradient method from smooth optimization.

It takes the directions of opposite subgradients as search directions and uses the pre-fixed step length in each iteration. The subgradient method is easy to implement, but it converges slowly. In contrast, the bundle-type method is a more efficient method; however in the same time, it is more complicated. In each iteration, the search direction is found by solving a quadratic programming problem and the step length is calculated by the line search.

The quasisecant method is one of the typical deterministic methods for solving nonsmooth optimization problems. The main idea of this method is similar to that of the bundle-type method. However, instead of using subgradients, the quasisecant method uses quasisecants to approximate the subdifferential. In the quasisecant method, an overestimation of the objective function is constructed instead of the lower piecewise linear approximation as in the bundle method.

The subgradient and bundle-type methods are efficient and robust for nonsmooth convex optimization problems, but they are inefficient in finding global optimal solutions for nonsmooth nonconvex optimization problems, because they can be easily trapped in a stationary point of the objective function.

Metaheuristic methods, which are the counterpart of deterministic methods, are designed for global optimization. They are also called stochastic methods since they may use a special stochastic procedure to generate points in the search process. The development of metaheuristic methods began in the 1960s and many algorithms have been developed thereafter. Figure 0.1 illustrates the timeline of various original metaheuristic methods. One can see from the figure that evolutionary algorithms were mainly introduced in 1960s; however, the research was limited to theoretical considerations, have not been applied to solve practical problems. From the early of 1990s to date, thanks to the dramatic development of computer technologies, metaheuristic methods can now be extensively applied to solve engineering problems.

Metaheuristic methods can be classified based upon background. Some methods simu-

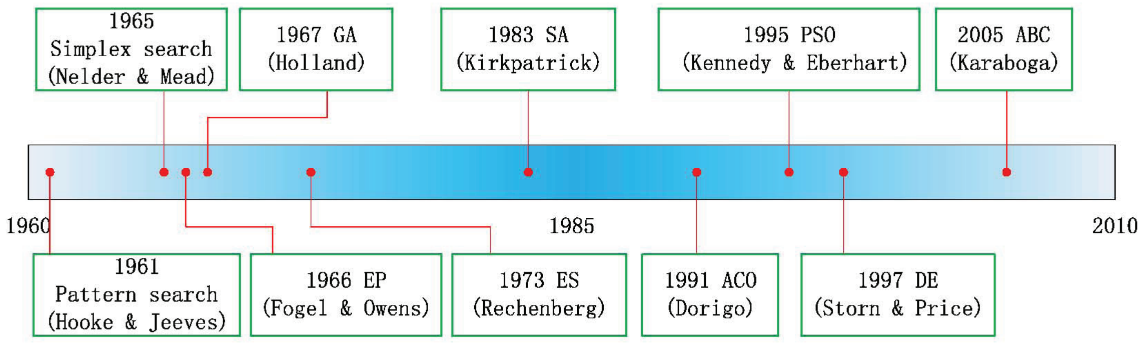


Figure 0.1.: Timeline of various original metaheuristic methods

late the evolutionary process of the biological world. The crossover, mutation and selection of biological evolution processes are simulated by binary code or real-number code. Metaheuristic methods of this type are called evolutionary algorithms, which include the evolution strategy, evolutionary programming, genetic algorithm and differential evolution. Some algorithms simulate the behavior of insects and birds. Metaheuristic methods of this type includes the artificial bee colony algorithm which simulates honey process of bees, the ant colony algorithm which simulates the foraging process of ant colony, the particle swarm optimization which simulates the behaviors of birds swarm. Another type of metaheuristic methods simulate physical processes, such as the simulated annealing.

Unlike most deterministic methods, which are good at local search but inefficient at global search, metaheuristic methods are good at global exploration but inefficient at local exploitation. Therefore, it is natural to yield the idea of combining both the advantages of deterministic and metaheuristic methods, but avoid the disadvantages of them. This idea gives rise to a new type of method for global optimization: the hybrid method. Actually, a number of hybrid methods have already been developed over the last three decades, but most of them are designed to solve global optimization problems with smooth objective functions.

In this thesis, we design two hybrid optimization methods for global optimization problems with nonsmooth nonconvex objective functions and apply them to solve some problems from applications. The thesis consists of two parts. In the first part we present hybrid methods for solving (nonsmooth) nonconvex optimization problems. In the second part we apply these methods to solve the system of nonsmooth equations, the molecular conformation and sensor localization problems.

First, we design a hybrid method based upon the quasisecant method. This method consists of two main phases: local search and global search. The local search phase is designed based upon the quasisecant method. It is applied to calculate local minimizers of nonsmooth nonconvex optimization problems. The global search phase is an implementation of a metaheuristic strategy. This strategy intends to find the boundary of local basins, which in return, enables the search to escape from local minimizers. More specifically, in a certain local minimizer, the objective function is approximated on spheres with different radii and an approximate local minimizer on each sphere is found. Some points are then selected from those approximate local minimizers by a special procedure. These points are located in different basins of the objective function. After this, the quasisecant method is applied starting from these points to find a set of local minimizers. The best local minimizer among all local minimizers is accepted as a new approximation to a global minimizer and so on. It is proved that this algorithm converges to the global optimal solution if the number of local minimizers of the objective function is finite and they are isolated. Numerical results are presented to demonstrate the efficiency of the proposed algorithm.

Our second hybrid method is based on the combination of the genetic algorithm and the derivative-free Hooke Jeeves method. The genetic algorithm, as a typical metaheuristic method, is good at globally searching a space, but it requires a huge computational effort since it is population-based and converges slowly. Furthermore, the accuracy of the genetic algorithm is not high. Therefore, we design an acceleration operator based upon the Hooke Jeeves method. This operator can pick some suitable points from the generations of the genetic algorithm and do local search starting from these points. In this way, the acceleration operator improves the convergent rate and the accuracy of the genetic algorithm. We apply this hybrid method to solve constrained optimization problems. The constraints in these problems are handled by the penalty function method. We consider two different penalty function models, quadratic penalty function and exact penalty function. The numerical performances of the proposed hybrid method on these two models are compared. We also compare the proposed hybrid method with other constrained optimization methods using the results of the numerical experiments.

In the second part of the thesis we first apply the quasisecant method to solve the system of nonsmooth equation systems. Then the hybrid global optimization method is applied to solve the molecular conformation and sensor localization problems.

First, an algorithm based upon the quasisecant mtehod for solving the system of nonsmooth equations is proposed . The system of nonsmooth equations is transformed into a nonsmooth optimization problem with a zero minimal objective function value. Then, the quasisecant method is applied to solve the nonsmooth optimization problem. Two different nonsmooth optimization models are studied and the convergent properties of the algorithm are investigated. The numerical performance of the proposed algorithm is presented.

We then apply the proposed hybrid global optimization method to solve the molecular conformation and sensor localization problems. The minimum-energy configuration of a small cluster of atoms, molecules or ions is known as the molecular conformation problem. It is a central problem in the study of cluster statics, or the topography of a potential energy function in an internal configuration space. From a mathematical point of view, molecular conformation problem is a difficult global optimization problem which does not yield easily either to discrete or continuous optimization methods. We solve the molecular conformation problems with number of atoms ranging from 2 to 23, i.e., the number of variables of the molecular conformation problems are from 6 to 69 with a increment of 3. The optimal solutions are achieved for all the problems, better optimal solutions are even obtained for problems with 8 and 22 atoms.

The sensor localization problem is the core subject of the wireless sensor network. The wireless sensor network is built of "nodes" from a few to several hundreds or even thousands, where each node is connected to one or many other sensors. The wireless sensor network presents novel tradeoffs in system design. On the one hand, the low cost of nodes facilitates massive scale and highly parallel computation. On the other hand, each node is likely to have limited power, limited reliability and only local communication with a modest number of neighbors. These application contexts and potential massive scale make it unrealistic to rely on careful placement or uniform arrangement of sensors. It is still impossible to localize each sensor by GPS because of the expensive cost and the limited power and memory of sensors. This leads to the area of the sensor localization problem which intends to localize the position of each sensor in a network by giving measured distances between the connective pairs of sensors. We solve the sensor localization problems with number of sensors from 10 to 50 (with a increment of 10), 100 to 500 (with a increment of 100), 1000 and 2000, respectively. The number of variables for these problems is up to 4000. The proposed hybrid global optimization method successfully solves all the sensor localization problems with high accuracy (up to 10^{-8}) of the results, which verifies the efficiency and reliability of this method.

Outline of the thesis

The remainder of the thesis is organized as follows;

In Chapter 1, we briefly introduce some basic definitions and related theorems on nonsmooth analysis and review some conventional methods for nonsmooth optimization.

In Chapter 2, we present a nonsmooth optimization method, i.e., the quasisecant method. Some results of numerical experiments are presented and analyzed.

In Chapter 3, we present a hybrid method for nonsmooth global optimization problems. Some academic benchmarks for global optimization are used to test this method and its results are compared with some other global optimization methods.

In Chapter 4, we propose another hybrid method for solving constrained global optimization problems. The numerical performance of this method for is evaluated using constrained optimization problems.

In Chapter 5, we propose a solver for the system of nonsmooth equations. This solver is applied to solve some systems of nonsmooth equations appearing in the bilevel programming and nonlinear complementarity problems.

In Chapter 6, the hybrid method developed in Chapter 3 is applied to solve the molecular conformation and sensor localization problems.

Chapter 7 contains some concluding remarks and recommendations for the future work.

Chapter 1.

Literature review

1.1. Nonsmooth analysis

Nonsmooth analysis studies the properties of nondifferentiable functions. Convexity is an important concept in nonsmooth analysis. Convex set and convex function are all well-defined and their geometrical interpretations are clear and apprehensible. In smooth optimization, we always start the research by convex cases and then generate it to nonconvex ones. This process is still applied to nonsmooth analysis and nonsmooth optimization. Thus, our strategies to study nonsmooth analysis are pretty apparent, from smoothness to nonsmoothness (from known to unknown) and from convexity to nonconvexity (from simple to hard). Actually, these are some general strategies in mathematical studying.

In optimization theory, the meaning of differentiation is to locally linearize the given differentiable function in the sense of the hyperplane generated by the the gradient is a tangent plane of the graph of the function. For convex function, these linearizations are always lower approximations. We have a lower piecewise linear approximation by taking a maximum over the linearizations defined at several points and by taking the maximum over all points we have the original convex differentiable function. These ideas are generalized for nonsmooth convex functions by defining the concepts of subgradient and subdifferential. A *subgradient* at a fixed point is a vector which is a lower approximation to the function; the set of all subgradients at that point is called *subdifferential.* These new concepts made it possible to obtain the same approximation properties for nonsmooth optimization as in the smooth cases.

It was a long step in nonsmooth analysis to generalize the subgradient, which was developed in convex cases, to nonconvex cases. As we have discussed, a linear piecewise lower approximation can be generated for convex function, but this is not the case for nonconvex function. For nonconvex function, we can think of differentiation as convexification. In this way, the basic ideas are the same as convex analysis, but the results are much more theoretical. In some researches, instead of subgradients, secants or quasisecants are considered. This may approximate nonconvex function in some sense, but the approximation is rough.

Contents of this section are mainly based upon the works of Rockafellar [100] and Clarke [25].

1.1.1. Notations and basic definitons

We discuss problems in $\mathbb{R}^n$, where $\mathbb{R}^n$ is the n-dimensional real Euclidean space. All the vectors are defined as column vectors. For vectors x and y, we denote by $x^T y$ (or $\langle x, y\rangle$) as the usual inner product, i.e.,

$$x^T y = \sum_{i=1}^{n} x_i y_i \left(\text{or } \langle x, y\rangle = \sum_{i=1}^{n} x_i y_i \right);$$

and denote by $\| x \|$ the Euclidean norm, i.e.,

$$\| x \| = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}.$$

The open ball with center x and radius $\lambda > 0$ is denoted by $B(x, \lambda)$, i.e.,

$$B(x, \lambda) = \{z \in \mathbb{R}^n \| \parallel x - z \parallel < \lambda\}.$$

We denote by $[x, y]$ the closed line segment joining x and y, i.e.,

$$[x, y] = \{z \in \mathbb{R}^n | z = \lambda x + (1 - \lambda)y \text{ for } 0 \leq \lambda \leq 1\};$$

and by (x, y) the corresponding open line segment. A set $C \subset \mathbb{R}^n$ is called convex if $[x, y] \subset C$ for all x and y belonging to C. For a series of points $x_1, x_2, \cdots, x_k$ in $\mathbb{R}^n$, x is called a linear combination of x_is if

$$x = \sum_{i=1}^{k} \lambda_i x_i,$$

where $\lambda_i \in \mathbb{R}$, $i = 1, 2, \cdots, k$; if, furthermore, $\lambda_i \geq 0$ $i = 1, 2, \cdots, k$ and $\sum_{i=1}^{k} \lambda_i = 1$, then x is called a convex combination of x_is. The convex hull of a set $C \subset \mathbb{R}^n$, denoted by conv$C$, is the set of all convex combination of points in C. In other words, conv$C$ is the smallest convex set containing C. C is convex if and only if C and conv$C$ coincide. The closure of C is denoted by cl$C$.

The set C is said to be a cone if it contains all positive multiples of its elements, i.e., $x \in C$ and $\lambda > 0$ imply that $\lambda x \in C$.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *convex* if

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y),$$

where x and y are in $\mathbb{R}^n$ and $\lambda \in [0, 1]$. If strict inequality holds for $x \neq y$ and $\lambda \in (0, 1)$, then f is called *strictly convex*. A function $f : \mathbb{R}^n \to \mathbb{R}$ is *locally Lipschitz* with constant K

at $x \in \mathbb{R}^n$ if there exists some $\varepsilon > 0$ such that

$$|f(y) - f(z)| \leq K \parallel y - z \parallel \text{ for all } y, z \in B(x; , \varepsilon).$$

Furthermore, f is *positively homogeneous* if

$$f(\lambda x) = \lambda f(x)$$

for all $\lambda \geq 0$ and *subadditive* if

$$f(x + y) \leq f(x) + f(y)$$

for all x and y in $\mathbb{R}^n$. It is worth to note that positively homogeneous and subadditive imply convexity.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *upper semicontinuous* at x if for every sequence $\{x_i\}$ converging to x, we have

$$\limsup_{i \to \infty} f(x_i) \leq f(x);$$

and *lower semicontinuous* if

$$f(x) \leq \liminf_{i \to \infty} f(x_i).$$

Note that an upper and lower semicontinuous function is continuous.

For a nonsmooth function f, we denote by Ω_f the set of points where the function f is not differentiable, i.e.,

$$\Omega_f = \{x \in \mathbb{R}^n | f \text{ is not differentiable at } x\}.$$

1.1.2. Convex analysis

The theory of nonsmooth analysis is based upon convex analysis. For this reason, we first survey some basic definitions and theorems of convex analysis. Then we review the definition of subgradient and subdifferential of nonsmooth convex functions and present some basic results. Contents of this subsection are mainly cited from Rockafellar [100], further reading refers to Roberts and Varbery [116].

Definition 1.1.1 The *directional derivative* of f at x in the direction $v \in \mathbb{R}^n$ is defined by

$$f'(x;v) = \lim_{t \downarrow 0} \frac{f(x+tv) - f(x)}{t}.$$

If f is differentiable at x, then the directional derivative, which exists in every direction $v \in \mathbb{R}^n$, is a linear function of v and we have the relation

$$f'(x;v) = \nabla f(x)^T v,$$

where $\nabla f(x) \in \mathbb{R}^n$ is the gradient vector of f at x.

Theorem 1.1.1 Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function with a Lipschitz constant K at $x \in \mathbb{R}^n$. Then

(i) the directional derivative in each direction $v \in \mathbb{R}^n$ exists and satisfies

$$f'(x;v) = \inf_{t>0} \frac{f(x+tv) - f(x)}{t},$$

(ii) the function $v \longmapsto f'(x;v)$ is positively homogeneous and subadditive on $\mathbb{R}^n$ with

$$|f'(x;v)| \leq K \parallel v \parallel,$$

(iii) $f'(x;v)$ is upper semicontinuous as a function of $(x;v)$ and Lipschitz with constant K as a function of v on $\mathbb{R}^n$,

(iv) $-f'(x;v) \leq f'(x;v)$.

Definition 1.1.2 The *subdifferential* of a convex function $f : \mathbb{R}^n \to \mathbb{R}$ at $x \in \mathbb{R}^n$ is the set

$$\partial_c f(x) = \{\xi \in \mathbb{R}^n |\ f(x') \geq f(x) + \xi^T(x' - x) \text{ for all } x' \in \mathbb{R}^n\}.$$

Each element $\xi \in \partial_c f(x)$ is called a *subgradient* of f at x.

In the following we present the relationship between subdifferential and the ordinary directional derivative. As we shall see, it is enough to know either of the concepts, the other can be computed from that.

Theorem 1.1.2 Let $f : \mathbb{R}^n \to \mathbb{R}$ be convex. Then at every x we have

(i) $f'(x;v) = \max\{\xi^T v |\ \xi \in \partial_c f(x)\}$ for all $v \in \mathbb{R}^n$,

(ii) $\partial_c f(x) = \{\xi \in \mathbb{R}^n |\ f'(x;v) \geq \xi^T v$ for all $v \in \mathbb{R}^n\}$,

(iii) $\partial_c f(x)$ is nonempty, convex and compact set such that $\partial_c f(x) \subset B(0;K)$, where K is the Lipschitz constant of f at x.

(iv) the point-to-set mapping $\partial_c f(\cdot) : \mathbb{R}^n \to P(\mathbb{R}^n)$ is upper semicontinuous, i.e., if $y_i \to x$ and $\xi_i \in \partial_c f(y_i)$ for each i, then each accumulation point ξ of $\{\xi_i\}_{i=1}^{\infty}$ is in $\partial_c f(x)$.

In smooth case, the gradient at a certain point is unique. However, in nonsmooth case, the subgradient at a nondifferentiable point is not unique . Thus, subgradient is a generalization of gradient, while subdifferential is a generalization of classical derivative.

Theorem 1.1.3 If $f : \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable at $x \in \mathbb{R}^n$, then

$$\partial_c f(x) = \{\nabla f(x)\}.$$

Using subdifferential and subgradient, we can present a representation to a convex function.

Theorem 1.1.4 If $f : \mathbb{R}^n \to \mathbb{R}$ is convex then for all $y \in R^n$

$$f(y) = \max\{f(x) + \xi^T(y - x) |\, x \in R^n, \xi \in \partial_c f(x)\}.$$

Theorem 1.1.4 is important for nonsmooth optimization. We can use piecewise linear function to approximate convex function. But directly applying Theorem 1.1.4 is not easy, since we cannot calculate all the subdifferential. However, we can calculate some elements of the subdifferential to construct an underestimation.

1.1.3. Nonsmooth differential theory

The definitions and results of subgradient and subdifferential in the previous subsection are all given for convex functions. In this subsection we generalize those definitions to nonconvex Lipschitz continuous functions. According to Theorem 1.1.1 the directional derivative $f'(x, v)$ always exists when the function f is convex. This derivative may not exist when f is nonconvex. Moreover, for nonconvex functions this derivative needs not to be subadditive. Therefore the notion of the directional derivative is generalized to preserve such properties.

According to Theorem 1.1.1, the directional derivative only exists when function f is convex. That is why we have to generalize the directional derivative for nonconvex Lipschitz continuous function. After that, we can generalize other concepts analogously. The generalization can be done in many ways, we review the approach of Clarke [25] in finite dimensional case.

Generalization of derivative

Definition 1.1.3 (Clarke). Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at a point $x \in \mathbb{R}^n$. The *generalized directional derivative* of f at x in the direction of $v \in \mathbb{R}^n$ is defined by

$$f^\circ(x;v) = \limsup_{\substack{y \to x \\ t \downarrow 0}} \frac{f(y+tv) - f(y)}{t}.$$

The following summarizes some basic properties of the generalized directional derivative.

Theorem 1.1.5 Let f be locally Lipschitz at x with constant K. Then

(i) the function $v \longmapsto f^\circ(x;v)$ is positively homogeneous and subadditive on $\mathbb{R}^n$ with

$$f^\circ(x;v) \leq K \parallel v \parallel,$$

(ii) $f^\circ(x;v)$ is upper semicontinuous as a function of $(x;v)$ and Lipschitz with constant K as a function of v on $\mathbb{R}^n$,

(iii) $f^\circ(x;-v) = (-f)^\circ(x;v)$.

Given the definition of generalized directional derivative, we are ready to generalize the subdifferential to nonconvex Lipschitz functions.

Definition 1.1.4 (Clarke). Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at $x \in \mathbb{R}^n$. Then the *generalized subdifferential* of f at x is the set

$$\partial f(x) = \{\xi \in \mathbb{R}^n |\ f^\circ(x;v) \geq \xi^T v \text{ for all } v \in \mathbb{R}^n\}.$$

Each element $\xi \in \partial f(x)$ is called a *generalized subgradient* of f at x.

Sometimes, we drop the word "generalized" if there is no confusion to do so. The following are some basic properties of generalized subdifferential.

Theorem 1.1.6 Let f be locally Lipschitz at x with constant K. Then

(i) $\partial f(x)$ is a nonempty, convex, compact set such that $\partial f(x) \subset B(0; K)$,

(ii) $f^\circ(x; v) = \max\{\xi^T v |\ \xi \in \partial f(x)\}$ for all $v \in \mathbb{R}^n$,

(iii) the mapping $\partial f(\cdot) : \mathbb{R}^n \to P(\mathbb{R}^n)$ is upper semicontinuous.

It can be proved that if f is locally Lipschitz continuous and differentiable at x, then we have $\nabla f(x) \in \partial f(x)$. Furthermore, if f is continuously differentiable at x, then we have $\partial f(x) = \{\nabla f(x)\}$. This implies that the subdifferential is a generalization of the classical derivative. The following theorem shows that the subdifferential for Lipschitz function is a generalization of the subdifferential for convex function.

Theorem 1.1.7 If the function $f : \mathbb{R}^n \to \mathbb{R}$ is convex, then

(i) $f'(x; v) = f^o(x; v)$ for all $v \in \mathbb{R}^n$ and

(ii) $\partial_c f(x) = \partial f(x)$.

Subdifferential calculus

In this subsection, we provide a chain rule for the calculation of generalized subdifferential. Note that this rule is a generalization of classical chain rule for differentiable function. However, in this case we have to content with inclusions instead of equalities. If given the property of regularity, we can sharpen our rules by turning the inclusions to equalities.

Theorem 1.1.8 (Chain Rule). Let $h : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}$ be functions such that each component function $h_i : \mathbb{R}^n \to \mathbb{R},\ i = 1, \ldots, m$ is locally Lipschitz at $x \in \mathbb{R}^n$ and g is locally Lipschitz at $h(x) \in \mathbb{R}^m$. Then the composite function $f = g \circ h$, $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz at x and

$$\partial f(x) \subset \text{conv}\{\sum_{i=1}^{m} \alpha_i \xi_i |\ \xi_i \in \partial h_i(x) \text{ and } \alpha \in \partial g(h(x))\}. \tag{1.1}$$

Addition, Products and Quotients rules can be concluded from Chain rule. For example, by defining $h = (h_1, h_2)^T : \mathbb{R}^2 \to \mathbb{R}^2$, where $h(x_1, x_2) = x_1$, $h_2(x_1, x_2) = x_2$; $g : \mathbb{R}^2 \to \mathbb{R}$, where $g(h_1, h_2) = h_1 * h_2$ and $f = g \circ h$, we can detect

$$\partial f(x) \subset x_2 \partial h_1(x) + x_1 \partial h_2(x) = h_2(x)\partial h_1(x) + h_1(x)\partial h_2(x),$$

which is coincidence with the products rule.

Note that the Relationship (1.1) just gives a necessary condition of $\partial f(x)$. We need regularity property to sharpen the rule by turning the inclusion to equality.

Definition 1.1.5 The function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *regular* at $x \in \mathbb{R}^n$ if for all $v \in \mathbb{R}^n$ the directional derivative $f'(x; v)$ exists and

$$f'(x; v) = f^\circ(x; v).$$

The following are some sufficient condition for f to be regular.

Theorem 1.1.9 Let f be Lipschitz at x. Then f is regular at x if

(i) f is continuously differentiable at x,

(ii) f is convex,

(iii) $f = \sum_{i=1}^{m} \lambda_i f_i$, where $\lambda_i > 0$ and f_i is regular at x for each $i = 1, \ldots, m$.

Given the definition and sufficient condition of regular, we can have a compensation of the chain rule.

Theorem 1.1.10 Equality holds in the Relationship (1.1) if any one of the following additional hypotheses is valid:

(i) The function g is regular at $h(x)$, each h_i is regular at x and $\alpha_i \geq 0$ for all $i = 1, \ldots, m$. Then also f is regular at x.

(ii) The function g is regular at $h(x)$ and h_i is continuously differentiable at x for all $i = 1, \ldots, m$.

(iii) The case $m = 1$ and g is continuously differentiable at $h(x)$.

1.1.4. Nonsmooth optimization theory

In this subsection, we discuss optimality conditions for nonsmooth optimization. As everyone knows, in smooth optimization, the first order necessary condition for x^* to be a local minimizer is $\nabla f(x^*) = 0$. In the following we generalize this necessary condition to nonsmooth optimization. Contents of this subsection are cited from Studniarski [110] and Schirotzek [104].

Theorem 1.1.11 If $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz at x and attains its local minimum at x, then

(i) $0 \in \partial f(x)$, and

(ii) $f^o(x; v) \geq 0$ for all $v \in \mathbb{R}^n$.

In Theorem 1.1.11, the condtion (i) $0 \in \partial f(x)$ is a direct generalization of the first order necessary condition in smooth optimization. But this condition is not easy to verify in practical numerical computation, since we cannot calculate the whole subdifferential. It is well known that if f is a convex function, then local minimizer implies global minimizer, this result is generalized in the following theorem.

Theorem 1.1.12 If $f : \mathbb{R}^n \to \mathbb{R}$ is convex, then the following conditions are equivalent:

(i) f attains its global minimum at x,

(ii) $0 \in \partial_c f(x)$,

(iii) $f'(x; v) \geq 0$ for all $v \in \mathbb{R}^n$.

1.2. Nonsmooth optimization methods

The subgradient and bundle methods are two of the most important methods for solving nonsmooth optimization problems. Both of them are based on the assumption that only the objective function value and one arbitrary subgradient at each point are available. These assumptions have been proved to be quite natural in practice. Note that the objective function of the nonsmooth optimization problem could be nonconvex.

Subgradient methods were mainly developed in the Soviet Union and an excellent overview can be found in Shor[107]. The idea of the subgradient method is directly extended from s-mooth optimization. It takes the opposite subgradient vector as the next search direction at the current iteration point, and applys step sizes which are supplied in advance. This simple idea generates two critical questions: i) how can we choose those step sizes and ii) is there any implementable stopping criterion? Several different proposals have been given to answer these questions, but the lack of an implementable stopping criterion is still the main hand-icap of subgradient methods. Also, the subgradient method is not a descent method, since the opposite direction of an arbitrary subgradient may not yield a decrease of the objective function. Because of this inherent drawback of subgradient, the convergence of subgradient method is hard to be achieved, even it is achieved, the convergent rate is only linear or even less than that.

In recent years, the bundle method becomes the most promising approach to solve non-smooth optimization problems. Because the subdifferential of a nonsmooth function is a set and it is almost impossible to calculate numerically, it is difficult to compute a descent direction from this set. Thus, approximating subdifferential using the previous subgradi-ent information becomes a practical approach. ε-steepest descent methods, the pioneering bundle method, was develop in [69] and it is based on the conjugate subgradient method pro-vided in [68]. The handicap of ε-subgradient method is the priori choice of an approximation tolerance which controls the radius of the ball in which the bundle model is thought to be a good approximation to the objective function.

In [62], Kiwiel proposed a new bundle method basing on the classical cutting plane method developed by [24] and [57]. The basic idea in this generalized cutting plane method is to form a convex piecewise linear approximation to the objective function using the linearizations generated by subgradients. Kiwiel also presented two strategies to bound the number of stored subgradients: subgradient selection and subgradient aggregation. The disadvantage of this generalized cutting plane method is its sensitivity to the scaling of the objective function. Also the uncertain line search operations may require massive times of function evaluation.

The bundle trust region method [105, 135] and the Proximal Bundle method [63] are developed to overcome the drawbacks caused in the ε-subgradient method and generalized cutting plane method. It is a combination of the bundle idea and the classical trust region philosophy.

Another method to solve nonsmooth optimization problems is the quasisecant method provided by Adil [2]. The difference of the quasisecant method comparing with the bundle-type method is that the subgradient are replaced by the quasisecant in the approximation of subdifferential. The quasisecant method still approximates the objective function by a piecewise linear function, but this approximation may not be a lower approximation anymore. This relaxation makes the quasisecant method more promising for solving nonconvex nonsmooth problems.

In spite of different background of those methods provided above, from a practical point of view, they all generate the search direction at each iteration by solving a quadratic problem, only differing in technical details.

1.2.1. Subgradient methods

The history of the subgradient method starts from 1960s, The basic idea of the subgradient method is generalized from smooth optimization methods through replacing gradients by arbitrary subgradients. It is a simple method in the sense of its step sizes are set ahead and it does not need any complex calculation to obtain search direction. So it gets extensive applications since it was developed. Especially, it is one of the most favorite methods for engineers.

The general process of the subgradient method are as follows.

Algorithm 1.2.1 Subgradient Method

Step 0: Initialization, set an initial point x_0 and a set of step size $\{\alpha_i\}_{i=1}^{\infty}$, set $k := 0$.

Step 1: Stop criterion, check the condition for terminating loops, if it is satisfied, then stop and x_k is the optimal solution; otherwise, go to Step 2.

Step 2: Iteration, compute an arbitrary subgradient $g_k \in \partial f(x_k)$ and pick the step size α_k from step size set, then use iteration formula

$$x_{k+1} = x_k - \alpha_k \frac{g_k}{\| g_k \|} \tag{1.2}$$

to get the next point, let $k := k + 1$ and go to Step 1.

The Equation (1.2) is the iteration formulation for the subgradient method. The search direction is $d_k = -g_k / \| g_k \|$, where $g_k \in \partial f(x_k)$ is an arbitrary subgradient at x_k, α_k is the step size which is fixed ahead. In order to guarantee the convergence, α_ks have to satisfy some conditions. The following are some typical step size rules.

- Constant step size: $\alpha_k = \alpha$, where α is a small positive constant.

- Square summable but not summable: the step sizes satisfy

$$\alpha_k \geq 0, \quad \sum_{k=1}^{\infty} \alpha_k = \infty, \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty.$$

  One typical example is $\alpha_k = a/(b+k)$, where $a > 0$ and $b \geq 0$.

- If the optimal value f^* is known, then the step size rule is

$$\alpha_k = \lambda \frac{f(x_k) - f^*}{\| g_k \|},$$

  where $\lambda \in [0, 2]$.

We conclude this subsection by some analysis of advantages and disadvantages of the subgradient method.

Advantages of subgradient method

+ The process of the subgradient method is simple. One just needs to calculate a subgradient vector at each iteration.

+ Since the step size sequence is fixed ahead, no line search process is needed in the subgradient method, which makes it easy to implement.

+ Because of the clear and simple process of the subgradient method, it is widely used in engineering applications.

Disadvantages of subgradient method

- The subgradient method is not a descent method, since the search direction which is opposite to a arbitrary subgradient vector may not yield a descent direction.

- Choosing step size rule is a big challenge for the subgradient method and an implementable stopping criterion is still hard to obtain.

- The convergence rate for the subgradient method is not better than linear convergence rate.

1.2.2. Bundle methods

Bundle methods, which attract plenty of attention in recent years, is another category of nonsmooth optimization methods. The history of bundle methods originates from the cutting plane idea, which was developed independently in [24] and [57]. The basic idea of cutting place method is to use a piecewise linear function to approximate the objective function. By now, there are many kinds of bundle methods, such as the ε-subgradient method, generalized cutting place method, proximate bundle method and bundle trust region method. In order to investigate the relationship between those methods, we first present a general structure of bundle method and then inspect the existing bundle methods one by one. To make the discussion simple, we suppose that the objective function $f(x)$ in this section is a convex function.

A general structure of bundle methods

Suppose that we are now at the iteration k, what we have are the current point x_k, objective function value $f(x_k)$ and one subgradient $\xi_k \in \partial f(x_k)$. Suppose that we also stored a set of points and their subgradients from previous iterations, these points are called trial points.

In numerical optimization, what we have to do in every iteration is to find a search direction d_k which can decrease the objective function value. In other word, we need to solve the following *Descent Direction Finding*(DDF) problem,

$$\text{(DDF)} \quad \begin{cases} \text{Minimize} & f(x_k + d) - f(x_k) \\ \text{Subject to} & \| d \| \leq 1. \end{cases} \tag{1.3}$$

For smooth optimization problems, through the first order Taylor's expansion and second order Taylor's expansion, we can solve the corresponding Problem (DDF1.3) and obtain the *steepest descent direction*

$$d_k = -\nabla f(x_k) / \| \nabla f(x_k) \|$$

and the *Newton's direction*

$$d_k = -H(x_k)^{-1}\nabla f(x_k),$$

respectively. However, for nonsmooth optimization problems it becomes more complicated, since the subgradient, which is the counterpart of the gradient, is not unique and hard to calculate. We just know that the set of all subgradients, which is subdifferential, is a convex compact set, but we have no idea about the shape of this set, let alone to calculate it. So it is impossible to compute a descent direction directly from the subdifferential. Thus, instead of the Taylor's expansion, we use the subgradient linearization to approximate the objective function.

Definition 1.2.1 We define the ξ-*linearization* of f at y by

$$\bar{f}_\xi(x; y) = f(y) + \xi^T(x - y) \quad \text{for all} \quad x \in \mathbb{R}^n. \tag{1.4}$$

where $\xi \in \partial f(y)$; define *linearization error* for $\xi \in \partial f(y)$ by

$$\alpha(x, y) = f(x) - \bar{f}_\xi(x; y) \quad \text{for all} \quad x \in \mathbb{R}^n. \tag{1.5}$$

Note that due to the definition of subgradient for convex functions we have

$$\alpha(x; y) \geq 0 \quad \text{for all} \quad x, y \in \mathbb{R}^n.$$

We then describe a general bundle method that produces a sequence $\{x_k\}_{k=1}^{\infty} \subset \mathbb{R}^n$ converging to the global minimum of f, if it exists. We suppose that, in addition to the current iteration point x_k, we have some trial point $y_j \in \mathbb{R}^n$ (maybe from past iterations) and the corresponding subgradients $\xi_j \in \partial f(y_j)$ for $j \in J_k$, where J_k is the index set of those trial points.

The idea of bundle methods is to approximate f at the current point x_k from below by using a piecewise linear function, in other words, we replace f by a so-called *cutting plane model*

$$\hat{f}^k(x) = \max_{j \in J_k} \{f(y_j) + \xi_j^T (x - y_j)\}, \tag{1.6}$$

which can be written in the equivalent form

$$\hat{f}^k(x) = \max_{j \in J_k} \{f(x_k) + \xi_j^T (x - x_k) - \alpha_j^k\}, \tag{1.7}$$

where α_j^k are linearization errors

$$\alpha_j^k = \alpha(x_k, y_j) = f(x_k) - f(y_j) - \xi_j^T (x_k - y_j) \quad \text{for all } j \in J_k. \tag{1.8}$$

The following theorem can be proved easily.

Theorem 1.2.1 For all $x \in R^n$ and $j \in J_k$ we have

(i) $\hat{f}^k(x) \leq f(x)$,

(ii) $\hat{f}^k(y_j) = f(y_j)$ and

(iii) $\hat{f}^k(x) = \max_{j \in J_k} \{-\alpha(x_k; y_j) + \xi_j^T (x - x_k)\} + f(x_k)$.

Replacing $f(x_k + d)$ by $\hat{f}^k(x_k + d)$ in (DDF1.3), we get the following *Nonsmooth Descent Direction Finding*(NDDF) problem

$$\text{(NDDF)} \quad \begin{cases} \text{Minimize} & \hat{f}^k(x_k + d) - f(x_k) \\ \text{Subject to} & \| d \| \leq 1. \end{cases} \tag{1.9}$$

In fact, in the Problem (NDDF1.9), it is not easy to check the constraint $\| d \| \leq 1$. So we deal with it by adding a penalty term $1/2 d^T M_k d$ to objective function. this term can keep the approximation local enough and thus guarantee the existence of the solution. The regular and symmetric $n \times n$ matrix M_k intends to accumulate information about the curvature of f in a local area around x_k. It plays the role of the Hessian matrix in smooth optimization.

Furthermore, substitute (1.7) into (NDDF1.9), let $d = x - x_k$, we get another form of the Problem (NDDF 1.9),

$$\text{(NDDF)}' \quad \begin{cases} \text{Minimize} & \max\limits_{j \in J_k} \{\xi_j^T d - \alpha_j^k\} + \frac{1}{2} d^T M_k d \\ \text{Subject to} & d \in \mathbb{R}^n. \end{cases} \tag{1.10}$$

Note that the Problem (NDDF 1.10)′ is still a nonsmooth optimization problem. However, due to its piecewise linear nature, it can be rewritten as a (smooth) *Quadratic Descent Direction Finding* (QDDF) subproblem

$$\text{(QDDF)} \quad \begin{cases} \text{Minimize} & v + \frac{1}{2} d^T M_k d \\ \text{Subject to} & -\alpha_j^k + \xi_j^k d \le v \quad \text{for all } j \in J_k. \end{cases} \tag{1.11}$$

By dualizing we get an equivalent problem to (QDDF 1.11). That is to find multipliers λ_j^k where $j \in J_k$ solving the *Dual Descent Direction Finding*(DDDF) problem

$$\text{(DDDF)} \quad \begin{cases} \text{Minimize} & \frac{1}{2} \left[\sum\limits_{j \in J_k} \lambda_j \xi_j \right]^T M_k^{-1} \left[\sum\limits_{j \in J_k} \lambda_j \xi_j \right] + \sum\limits_{j \in J_k} \lambda_j \alpha_j^k \\ \text{Subject to} & \\ & \sum\limits_{j \in J_k} \lambda_j = 1 \\ & \lambda_j \ge 0. \end{cases} \tag{1.12}$$

Notice that in the Problem (DDDF 1.12) the sign has been changed and maximization has been converted to minimization. For computational reasons it might in some cases be more efficient to solve the Problem (QDDF 1.11) instead of the Problem (DDDF 1.12). The following theorem establishes the relationship between these two problems.

Theorem 1.2.2 The Problems (QDDF 1.11) and (DDDF 1.12) are equivalent, and they have unique solutions (d_k, v_k) and λ_j^k for $j \in J_k$, respectively, such that

$$d_k = -\sum_{j \in J_k} \lambda_j^k M_k^{-1} \xi_j, \tag{1.13}$$

$$v_k = -d_k^T M_k d_k - \sum_{j \in J_k} \lambda_j^k \alpha_j^k. \tag{1.14}$$

Based on the discussion presented above, we can design a general algorithm for bundle methods.

Algorithm 1.2.2 The General Bundle Method

Step 0: Initialization. Choose a starting point x_0, evaluate the objective function value $f(x_0)$ and a subgradient $\xi_0 \in \partial f(x_0)$. Let $M_0 = I, k = 1, J_0 = \varnothing$.

Step 1: Stopping criterion. Check the stopping criterion at point x_k. If the stopping criterion is satisfied, then stop iteration and x_k is the optimal solution.

Step 2: Updating. Update J_{k-1} and M_{k-1} to J_k and M_k, respectively.

Step 3: Descent direction. Solve the descent direction finding problem (QDDF 1.11) or the dual descent direction finding problem (DDDF 1.12) getting the next search direction d_k.

Step 4: Line search. Apply a line search method to compute a step size α_k and compute a new point

$$x_{k+1} = x_k + \alpha_k d_k.$$

Let $k := k + 1$ and go to step1.

Note that there are many subproblems need to be solved in this general bundle method. In fact, different methods to solve these subproblems yield different practical bundle methods. We list those subproblems as follows and the detailed discussion of how to solve those subproblems in certain algorithm will be presented later.

1. What kind of stopping criterion should we use? A direct extension of smooth optimization suggests that $0 \in \partial f(x_k)$ should be used. But this condition is still not easy to verify since the total calculation of the subdifferential is impossible. Thus some new calculable stopping criterion need to be exploited.

2. How to choose trial points y_j is a big problem in bundle method. The set of trial points can decide the quality of approximation of the objective function at x_k, which, as a result, verifies the direction we obtained is a good descent direction or not. Unfortunately, we do not know what kind of point can be a good trial point. And furthermore, the number of trial points still influence the descent direction finding problem. If there are too many trial points, the size of descent direction finding problem will become very large which makes the cost of calculation very expensive. So our aim is to use a smaller number of trial points to approximate objective function as accurate as possible. A simple approach to pick trial points is to directly consider the previous iteration points $x_i, i = 1, 2, \ldots, k-1$. Obviously, this strategy is rough since some of the iteration points are far from the current point x_k.

3. An efficient and executable line search method is essential in the bundle method. A simple idea is to successively double the step size until we get a satisfying decrease. A popular approach in many literatures is to expand the inexact line search method.

4. A null step is needed in the bundle method. We use a polytope to approximate the subdifferential at the current point and the search direction is obtained by solving the corresponding quadratic problem. However, impacted by the accuracy of approximation, the obtained search direction could not be a good descent direction, even not a descent direction, which means that the approximation needs to be improved. A null step, in this case, keeps the current point and accumulates more information around the current point in order to build a more accurate descent direction finding model. Generally, a

trial point will be generated after every iteration. If it is a null step, then this trial point is collected to build a better approximation of the subdifferential. Sometimes, large number of null steps are applied before a good approximation is constructed.

We now introduce some typical bundle method and point out their advantages and disadvantages. And at the same time, we analyze the relationship between those methods.

Cutting plane methods and their variations

According to the definition 1.2.1, we have

$$\begin{aligned} & \min_{d\in\mathbb{R}^n} \; f(x_k+d)-f(x_k) \\ \approx & \min_{d\in\mathbb{R}^n} \; \hat{f}^k(x_k+d)-f(x_k) \\ = & \min_{d\in\mathbb{R}^n} \max_{j\in J_k} \; \{-\alpha_j^k+\xi_j^k d\}. \end{aligned} \tag{1.15}$$

Then, due to the minmax construction, (1.15) can be transformed to find a solution $(d,v)\in\mathbb{R}^{n+1}$ to a linearly constrained smooth *Cutting Plane Model.*

$$\text{(CPM)} \quad \begin{cases} \text{Minimize} \quad v \\ \text{Subject to} \quad -\alpha_j^k+\xi_j^k d \le v \quad \text{for all} \quad j\in J_k. \end{cases} \tag{1.16}$$

There are two main drawbacks in the original cutting plane method. Firstly, the Problem (CPM 1.16) does not necessarily have a solution and secondly, the method generally attains rather poor convergence results in practice. To avoid these disadvantages, the following generalization was proposed by Kiwiel in [62]. By adding a regularizing penalty term $\frac{1}{2}\parallel d\parallel^2$ to the objective function of Problem (CPM 1.16),we have the *Generalized Cutting Plane*

Model (GCPM)

$$
\text{(GCPM)} \quad \begin{cases} \text{Minimize} & v + \frac{1}{2} \parallel d \parallel^2 \\ \text{Subject to} & -\alpha_j^k + \xi_j^k d \le v \quad \text{for all} \quad j \in J_k. \end{cases} \tag{1.17}
$$

By duality, Problem (GCPM 1.17) is equivalent to find multipliers λ_j^k ($j \in J_k$) solving the quadratic problem

$$
\text{(DGCPM)} \begin{cases} \text{minimize} & \frac{1}{2} \parallel \sum\limits_{j \in J_k} \lambda_j \xi_j \parallel^2 + \sum\limits_{j \in J_k} \lambda_j \alpha_j^k \\ \text{subject to} & \\ & \sum\limits_{j \in J_k} \lambda_j = 1 \\ & \lambda_j \ge 0 \quad \text{for all} \quad j \in J_k. \end{cases} \tag{1.18}
$$

If the multipliers λ_j^k solve the Problem (DGCPM 1.18), then we obtain the search direction

$$
d_k = -\sum_{j \in J_k} \lambda_j^k \xi_j.
$$

One advantage of the Problem (DGCPM 1.18) is that we do not need to choose the approximation tolerance ε_k, but this subproblem is rather sensitive to the scaling of the objective function (i.e. multiplication of f by a positive constant). It is also worth to note that the Problem (QGCPM 1.18) is a special form of the Problem (QDDF 1.11) with the choice

$$
M_k \equiv I.
$$

The conjugate subgradient method

The conjugate subgradient method is generalized from the idea of conjugate gradient method which constructs the next search direction based upon the previous search directions. This method was developed in [68] and [122].

We define a convex set as

$$S_k = \text{conv}\{-\xi_j |\, j \in J_k\}.$$

and calculate the search direction d_k as the projection of original point onto the set. In other words, we solve the following problem,

$$\begin{cases} \text{Minimize} & \| \, g \, \| \\ \text{subject to} & g \in S_k. \end{cases} \tag{1.19}$$

It is no doubt that S_k can also be expressed as

$$S_k = \{g \in \mathbb{R}^n |\, g = -\sum_{j \in J_k} \lambda_j \xi_j,\ \lambda_j \geq 0 \text{ for all } j \in J_k,\ \sum_{j \in J_k} \lambda_j = 1\}.$$

Thus, Problem (1.19) is equivalent to

$$\text{(CSM)} \quad \begin{cases} \text{Minimize} & \frac{1}{2} \, \| \sum\limits_{j \in J_k} \lambda_j \xi_j \, \|^2 \\ \text{Subject to} & \\ & \sum\limits_{j \in J_k} \lambda_j = 1 \\ & \lambda_j \geq 0 \quad \text{for all } j \in J_k. \end{cases} \tag{1.20}$$

This dual approach means that each of the previous subgradients $\xi_j \in \partial f(y_j)$ for $j \in J_k$ is treated as a subgradient at the current point x_k, in other words, the linearization error (1.8) was neglected. On the other hand, no curvature of the objective function was taken into account, i.e., the search direction fining problem (QDDF1.11) and (DDDF1.12) were applied with

$$M_k \equiv I \ \text{ and } \ \alpha_j^k \equiv 0.$$

The neglect of α_j^k has the effect that the cutting plane model $\hat{f}^k$ is a usable approximation to f only if the trial points y_j are close enough to x_k. For this reason, the choice of the index set

$J_k \subset \{1, \ldots, k\}$ is a crucial point of the algorithm. Several *subgradient selection strategies* to choose J_k were proposed, for instance, in [68, 90, 122], to keep the approximation local enough and limit the number of stored subgradients. In [68, 122] the authors, for the first time, introduced the *subgradient aggregation strategy*, which requires only a limited number of subgradients. The convergence of conjugate subgradient method has been proved in [36]. However, the numerical experiments have shown that the convergence of the conjugate subgradient method is rather slow in practice (see [70]).

ε-subgradient methods

The ε-subgradient method could be the first method being claimed as "bundle method". It was introduced in [67], where the idea was to combine the cutting plane method with the conjugate gradient method. The method was further developed in [71], where the subgradient aggregation strategy was used to limit the number of stored subgradients. Other contributions to the ε-subgradient method are [11], [12] and [109], etc.

Before the discussion of ε-subgradient method, we firstly recall some basic definitions and theorems of ε-subgradient, since ε-subgradient method is based on them.

Definition 1.2.2 Let f be any convex function finite at x. A vector ξ is called an ε*-subgradient* of f at x (where $\varepsilon > 0$) if

$$f(y) \geq f(x) + \xi^T(y - x) - \varepsilon \quad \text{for all} \quad y \in \mathbb{R}^n.$$

The set of all such ε-subgradient is called ε-subdifferential and denoted by $\partial_\varepsilon f(x)$.

The following summarizes some basic properties of the ε-subdifferential.

Theorem 1.2.3 Let $f : \mathbb{R}^n \to \mathbb{R}$ be convex. Then

(i) $\partial_0 f(x) = \partial_c f(x)$.

(ii) If $\varepsilon_1 < \varepsilon_2$, then $\partial_{\varepsilon_1} f(x) \subset \partial_{\varepsilon_2} f(x)$.

(iii) $f'_\varepsilon(x; v) = \max\{\xi^T v | \ \xi \in \partial_\varepsilon f(x)\}$ for all $v \in \mathbb{R}^n$.

(iv) $\partial_\varepsilon f(x) = \{\xi \in \mathbb{R}^n | f'_\varepsilon(x; v) \geq \xi^T v$ for all $v \in \mathbb{R}^n\}$.

(v) $\partial_\varepsilon f(x)$ is nonempty, convex and compact set such that $\| \ \xi \ \| \leq K$ for all $\xi \in \partial_\varepsilon f(x)$.

(vi) The mapping $\partial_\varepsilon f(\cdot) : \mathbb{R}^n \to \mathcal{P}(\mathbb{R}^n)$ is upper semicontinuous.

Theorem 1.2.4 Let $f : \mathbb{R}^n \to \mathbb{R}$ be convex with Lipschitz constant K at x. If $\varepsilon \geq 0$, then

$$\partial_c f(y) \subset \partial_\varepsilon f(x) \quad \text{for all} \quad y \in B(x; \varepsilon/2K).$$

From the Theorem1.2.4, we know that when y close enough to x, the ε-subdifferential at y can be considered as an approximation of the subdifferential at x. And furthermore, we can simply use the ε-subdifferential an approximation of the subdifferential at x. According to this point of view, we can explore the following ε-subgradient method.

In smooth optimization, in order to find a descent search direction, we approximate the objective function of Problem (DDF 1.3) by

$$f(x_k + d) - f(x_k) \approx \nabla f(x_k)^T d.$$

In this approximation point of view, the Problem (DDF 1.3) is equivalent to

$$\begin{cases} \text{Minimize} & \nabla f(x_k)^T d \\ \text{Subject to} & \| \ d \ \| \leq 1. \end{cases} \tag{1.21}$$

Solving Problem (1.21), we have the steepest descent direction $-\nabla f(x_k)/ \ \| \ \nabla f(x_k) \ \|$.

In nonsmooth case, we have a counterpart of problem (1.21)

$$\begin{cases} \text{Minimize} \quad f'(x_k, d) \\ \text{Subject to} \quad \| d \| \leq 1. \end{cases} \tag{1.22}$$

Since directional derivative is a supporting function of subdifferential, Problem (1.22) is equivalent to

$$\min_{\|d\| \leq 1} \max_{g \in \partial f(x_k)} g^T d, \tag{1.23}$$

and by the well-known Minmax Theorem, we can change the optimization order and have

$$\max_{g \in \partial f(x_k)} \min_{\|d\| \leq 1} g^T d. \tag{1.24}$$

If $g \in \partial f(x_k)$ and $g \neq 0$, then the solution of the latter minimization problem is $d = -g/ \| g \|$, so we have

$$\max_{g \in \partial f(x_k)} \min_{\|d\| \leq 1} g^T d = \max_{g \in \partial f(x_k)} g^T(-g/ \| g \|) = - \min_{g \in \partial f(x_k)} \| g \| .$$

Hence, for solving problem (1.22), we have to study the minimum-norm problem (which is uniquely solvable since $\partial f(x_k)$ is a nonempty closed convex set)

$$\begin{cases} \text{Minimize} \quad \| g \| \\ \text{Subject to} \quad g \in \partial f(x_k) \end{cases} . \tag{1.25}$$

Replacing the subdifferential in problem (1.25) by the ε-subdifferential, we have problem

$$\begin{cases} \text{Minimize} \quad \| g \| \\ \text{Subject to} \quad g \in \partial_\varepsilon f(x_k). \end{cases} \tag{1.26}$$

Based on the previous discussion, we are ready to design the ε-subgradient method[12].

Algorithm 1.2.3 The ε-Subgradient Method

Step 0: Select a vector x_0 such that $f(x_0) < \infty$, a scalar $\varepsilon_0 > 0$ and a scalar a, $0 < a < 1$.

Step 1: If $0 \in \partial_\varepsilon f(x_k)$, then stop iteration, x_k is the ε-optimal solution; otherwise, go to Step 2.

Step 2: Given x_k and $\varepsilon_k > 0$, set $\varepsilon_{k+1} = a^m \varepsilon_n$, where m is the smallest nonnegative integer such that $0 \notin \partial_{\varepsilon_{m+1}} f(x_k)$.

Step 3: Find a vector d_k such that

$$\max_{g \in \partial_{\varepsilon_{m+1}} f(x_k)} g^T d_k < 0.$$

Step 4: Set $x_{k+1} = x_k + \alpha_k d_k$, where $\alpha_k > 0$ is such that

$$f(x_k) - f(x_{k+1}) > \varepsilon_{k+1}.$$

Return to Step 2.

In fact, this algorithm is hard to realize in practical calculation since we do not know how to compute $\partial_{\varepsilon_k} f(x_k)$. So we have to approximate $\partial_\epsilon f(x_k)$ by some set which we can compute. We define an approximation of $\partial_{\varepsilon_k} f(x_k)$ by

$$G_k(\varepsilon_k) = \{\xi \in R^n |\, \xi = \sum_{j \in J_k} \lambda_j \xi_j, \sum_{j \in J_k} \lambda_j \alpha_j^k \leq \varepsilon_k, \lambda_j \geq 0, \sum_{j \in J_k} \lambda_j = 1\}.$$

It can be proved that set $G(\varepsilon_k)$ is convex and compact and

$$G(\varepsilon_k) \subset \partial_{\varepsilon_k} f(x_k).$$

Now we replace $\partial_{\varepsilon_k} f(x_k)$ by the set $G(\varepsilon_k)$ and change the objective function into and equivalent style $\frac{1}{2} \parallel g \parallel^2$ in (1.26), we have

$$\begin{cases} \text{Minimize} & \frac{1}{2} \parallel g \parallel^2 \\ \text{Subject to} & g \in G(\varepsilon_k) \end{cases}. \tag{1.27}$$

Due to the definition of $G(\varepsilon_k)$, this is equivalent to find the multipliers λ_j $(j \in J_k)$ solving the quadratic problem

$$\text{(SGM)} \begin{cases} \text{Minimize} & \frac{1}{2} \parallel \sum\limits_{j \in J_k} \lambda_j \xi_j \parallel^2 \\ \text{Subject to} & \\ & \sum\limits_{j \in J_k} \lambda_j \alpha_j^k \le \varepsilon_k \\ & \sum\limits_{j \in J_k} \lambda_j = 1 \\ & \lambda_j \ge 0 \text{ for all } j \in J_k. \end{cases}. \tag{1.28}$$

If the multiplies λ_j^k solve the Problem (SGM 1.28), then we obtain the search direction by

$$d_k = -\sum_{j \in J_k} \lambda_j^k \xi_j. \tag{1.29}$$

Comparing with the construction of $G(\varepsilon_k)$ and S_k, we can note that some attention to the linearization error (the Equation 1.8) was paid. The constraint $\sum_{j \in J_k} \lambda_j \alpha_j^k \le \varepsilon_k$ guaranteed that the trial points are not far from the current point x_k, thus a good approximation of subdifferential at x_k can be approached. Still, no curvature of the objective function was taken into account yet, in other words,

$$M_k \equiv I.$$

One drawback of this method is that the performance of the approximation is very sensitive to the choice of ε_k. Unfortunately, we do not have any good approach to chose it. There are still some other discussion of the ε-subgradient method, such as [12] where the author presented an exact representation of the ε-subdifferential for some special convex functions, but he did not present any numerical results.

Bundle trust region and proximal bundle methods

As discussed before, the penalty term $\frac{1}{2} \parallel d \parallel^2$ in the Problem (GCPM 1.17) plays a rule of restricting the radius of the bundle around x_k, i.e., norm of the direction d_k should not be too large. In order to obtain this aim, the bundle trust region method was developed by combining the bundle and trust region methods. It was first introduced in [105], and then developed in [72, 80].

Instead of adding the regularizing penalty term $\frac{1}{2} \parallel d \parallel^2$ to the objective function of the Problem (DDF 1.3), the idea of classical trust region methods is used to construct a restriction of the cutting plane model. Let $\sigma_k > 0$ and consider the following modification of the Problem (DDF 1.3)

$$\begin{cases} \text{Minimize} & \hat{f}^k(x_k + d) - f(x_k) \\ \text{Subject to} & \frac{1}{2} \parallel d \parallel^2 \leq \sigma_k, \end{cases} \tag{1.30}$$

we can have the *Bundle Trust Region Method*

$$\text{(BTRM)} \quad \begin{cases} \text{Minimize} & v \\ \text{Subject to} & \\ & -\alpha_j^k + \xi_j^T d \leq v \quad \text{for all} \quad j \in J_k \\ & \frac{1}{2} \parallel d \parallel^2 \leq \sigma_k. \end{cases} \tag{1.31}$$

Another approach to consider the role of term $\frac{1}{2} d^T M_k d$ is the so-called diagonal variable metric idea. A weighting parameter was added to the quadratic term of the objective function in Problem (QDDF 1.11) and Problem (DDF 1.3) in order to accumulate some second-order information about the curvature of f around x_k. Thus the variable metric matrix M_k took the diagonal form

$$M_k = \mu_k I,$$

where the weighting parameter $\mu_k > 0$. This method, named *Proximal Bundle Method*, was derived in [63] based upon the work of [99]. Proximal bundle method has a form

$$\text{(PBM)} \quad \begin{cases} \text{Minimize} & v + \frac{\mu_k}{2} \parallel d \parallel^2 \\ \text{subject to} & -\alpha_j^k + \xi_j^T d \le v \quad \text{for all} \quad j \in J_k. \end{cases} \tag{1.32}$$

Although originate from different ideas, The Problem (BTRM 1.31) and (PBM 1.32) have the following relationship:

(i) If (v_k, d_k) is an optimal solution of the Problem (BTRM 1.31) and $u(\sigma_k)$ is the corresponding Lagrange multiplier of the constraint

$$\frac{1}{2} \parallel d \parallel^2 \le \sigma_k,$$

then (v_k, d_k) is also an optimal solution of the Problem (PBM 1.32) for $\mu_k = \mu(\sigma_k)$.

(ii) If (v_k, d_k) is an optimal solution of the Problem (PBM 1.32), then it is also an optimal solution of the Problem (BTRM 1.31) for

$$\sigma_k = \frac{1}{2} \parallel d_k \parallel^2 .$$

The dual formulation of the Problem (PBM 1.32) is to find multipliers λ_j for $j \in J_k$ which solve the quadratic problem

$$
\text{(DPBM)}\begin{cases}
\text{minimize} & \frac{1}{2} \parallel \sum\limits_{j\in J_k} \lambda_j \xi_j \parallel^2 + \mu_k \sum\limits_{j\in J_k} \lambda_j \alpha_j^k \\
\text{Subject to} & \\
 & \sum\limits_{j\in J_k} \lambda_j = 1 \\
 & \lambda_j \geq 0 \quad \text{for all} \quad j \in J_k.
\end{cases} \tag{1.33}
$$

If the multipliers λ_j^k solve the the Problem (DPBM 1.33), then we obtain the search direction by

$$d_k = -\frac{1}{\mu_k} \sum_{j\in J_k} \lambda_j^k \xi_j.$$

Remark 1.2.1 The connection between the Problem (SGM 1.28) and (DPBM 1.33) are

(i) If λ^k ia an optimal solution of the Problem (SGM 1.28) and $\mu(\varepsilon_k)$ the corresponding Lagrange multiplier of the constraint

$$\sum_{j\in J_k} \lambda_j \alpha_j^k \leq \varepsilon_k,$$

then λ^k is also an optimal solution of the Problem (DPBM 1.33) for $\mu_k = \mu(\varepsilon_k)$.

(ii) If λ^k is an optimal solution of the Problem (DPBM 1.33), then it is also an optimal solution of the Problem (SGM 1.28) for

$$\varepsilon_k = \sum_{j\in J_k} \lambda_j^k \alpha_j^k.$$

Remark 1.2.2 We list some relationship between those methods we discussed above.

(i) Generalized cutting plane method (DGCPM 1.18) is a special case of the Problem (DPB-M 1.33) with $\mu_k = 1$.

(ii) The main difference between PB-methods (PBM 1.32) and BTR-methods (BTRM 1.31) consists in strategies for updating the weight μ_k.

(iii) PB-methods (PBM 1.32) and BTR-methods (BTRM 1.31) do not need potentially unreliable line searches, since the step size control is included in the control of the weight μ_k.

Titled proximal bundle methods

The most recent step in the development of bundle method was made in [64] basing upon the work of [113]. The main idea is to use the so-called titled cutting planes in order to get some second-order information and to employ some features of interior point methods. The matrix M_k was handled in the original diagonal form

$$M_k = \mu_k I,$$

with the safeguarded quadratic interpolation technique for updating μ_k.

Let $\kappa \in (0, 1]$, $\theta \in [0, 1-\kappa]$ and $\xi \in \partial f(y)$, we define the *titled linearization*

$$\bar{f}(x; y; \theta) = f(y) + (1-\theta)\xi^T(x-y) \quad \text{for all} \quad x \in \mathbb{R}^n. \tag{1.34}$$

Note that

$$\bar{f}(x; y; 0) = \bar{f}(x; y).$$

since $\bar{f}(x; y; \theta)$ need no longer be a lower approximation and the original linearization error may be negative, we define also a new linearization error by

$$\alpha(x; y; \theta) = \max\{f(x) - \bar{f}(x; y; \theta), \kappa\, \alpha(x; y)\}.$$

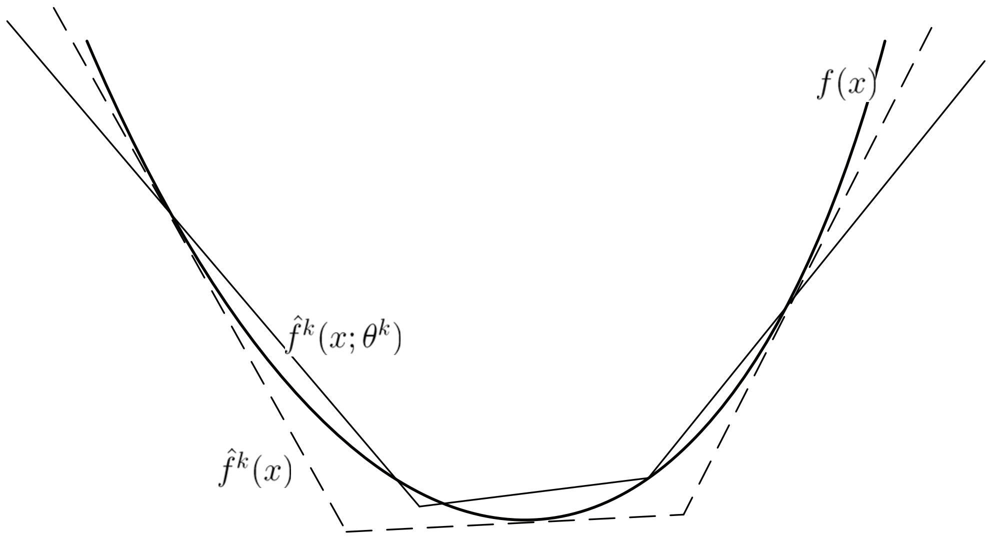


Figure 1.1.: The tilted cutting plane model

Evidently,

$$\alpha(x;y;\theta) \geq 0 \tag{1.35}$$

due to $\kappa > 0$. The following result gives a motivation to employ tilted approximations.

Lemma 1.2.1 If f is strictly convex and quadratic with the minimizer x^*, then

$$\bar{f}(x^*;y;\theta) \leq \bar{f}(x^*;y;\frac{1}{2}) = f(x^*) \quad \text{for all} \quad y \in \mathbb{R}^n \text{ and } \theta \in [0,\frac{1}{2}].$$

Now let $\xi \in \partial f(y)$ for $j \in J_k$ and let $\theta_j^k \in [0, 1-\kappa]$ for $j \in J_k$ be the corresponding tilting coefficients. Then for all $x \in R^n$ we define the following tilted approximation to f

$$\hat{f}^k(x,\theta^k) = \max_{j \in J_k}\{-\alpha(x_k;y_j;\theta_j^k) + (1-\theta_j^k)\xi_j^T(x-x_k)\} + f(x_k).$$

Due to Equality (1.34), $\hat{f}^k(x;\theta^k)$ is no longer a lower approximation of f. However, Inequality 1.35 ensures that

$$\hat{f}^k(x_k;\theta^k) \leq f(x_k).$$

To exploit the proximal bundle idea we replace the Problem (PBM 1.32) with

$$\text{(TPBM)} \quad \begin{cases} \text{Minimize} & v + \frac{\mu_k}{2} \parallel d \parallel^2 \\ \text{Subject to} & -\alpha(x_k; y_j; \theta_j^k) + (1-\theta_j^k)\xi_j^T d \leq v \quad \text{for all} \quad j \in J_k. \end{cases} \tag{1.36}$$

and the dual formulation of the Problem (TPBM 1.36) is to find multiplies λ_j^k $(j \in J_k)$ which solve the problem

$$\text{(DTPBM)} \quad \begin{cases} \text{minimize} & \frac{1}{2} \parallel \sum_{j \in J_k} \lambda_j (1-\theta_j^k)\xi_j \parallel^2 + \mu_k \sum_{j \in J_k} \lambda_j \alpha(x_k; y_j; \theta_j^k) \\ \text{Subject to} & \\ & \sum_{j \in J_k} \lambda_j = 1 \\ & \lambda_j \geq 0 \quad \text{for all} \quad j \in J_k. \end{cases} \tag{1.37}$$

If the multiplies λ_j^k solve the Problem (DTPBM1.37), then we obtain the search direction by

$$d_k = -\frac{1}{\mu_k} \sum_{j \in J_k} \lambda_j^k (1-\theta_j^k)\xi_j.$$

Note that if $\kappa = 1$, then $\theta_j^k = 0$ for all $j \in J_k$ and the method reduces to the proximal bundle method described previously. The tilted cutting plane is a real ”cutting plane“, since it cuts off a parts of the epigraph of f while the standard cutting plane cuts nothing. The numerical tests in [64] appear rather promising, but the question of how to choose the tilting parameters κ and θ_j^k is still unsolved.

1.3. Global optimization

Global optimization is a branch of applied mathematics and numerical analysis that deals with the optimization of a function or a set of functions according to some criteria, such as a set of bound or more general constraints [45]. The objective of global optimization is to find the globally best solution of models among a number of possible local optimal solutions. Formally, global optimization seeks global solution(s) of a constrained optimization model. It is now established that global optimization has ubiquitous applications in various areas, such as chemical engineering [8, 55], applied sciences [89], biotechnology [119], data analysis [17, 51], environmental management [14], risk management [13], and so on. A general model of global optimization problem can be formulated as follows.

$$\begin{cases} \text{Minimize} & f(x) \\ \text{Subject to} & g_i(x) \leq 0, \quad i = 1, 2, \cdots, l \\ & h_j(x) = 0 \quad j = 1, 2, \cdots, m \\ & x \in X, \end{cases} \tag{1.38}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is *objective function*, $g_i : \mathbb{R}^n \to \mathbb{R},\ i = 1, 2, \cdots, l$ are *inequality constraint functions*, $h_j : \mathbb{R}^n \to \mathbb{R},\ j = 1, 2, \cdots, m$ are *equality constraint functions*, and

$$X = \{x = (x_1, x_2, \cdots, x_n) \mid l_i \leq x_i \leq u_i \quad i = 1, 2, \cdots, n\}$$

is a box set, $l = (l_1, l_2, \cdots, l_n)^T$ and $u = (u_1, u_2, \cdots, u_n)^T$ are *lower boundary* and *upper boundary*, respectively. We define the *search space* of Problem (4.1) as

$$S = \{x \in \mathbb{R}^n | x \in X\},$$

furthermore, the *feasible search space* as

$$F = \{x \in S \mid g_i(x) \leq 0,\ i = 1, 2, \cdots, l, \quad h_j(x) = 0,\ j = 1, 2, \cdots, m\}.$$

Any point $x \in F$ is called a *feasible point*.

There are three different types of methods to solve global optimization problems, deterministic methods, metaheuristic methods and hybrid methods. They will be briefly introduced in the following subsections.

1.3.1. Deterministic methods

The main concept of deterministic global optimization methods is that in the generic procedure of solving global optimization problems, the next iteration does not depend on the outcome of a pseudo random variable. Such a method gives a fixed sequence of steps when the algorithm is repeated for the same problem. But there is not necessarily a guarantee to reach the optimum solution. Some excellent introductions of deterministic methods refer to [28, 46, 47].

The most important deterministic global optimization method is the branch-and-bound method [34, 102, 103, 127]. It consists of two parts, branching and bounding. Branching refers to successive partitioning (or subdivision) of the feasible domain, whereas bounding refers to the computation of lower and upper bounds of global optimum. In each iteration of the branch-and-bound method, the search region is divided into finitely many subregions, then the maximum lower bound and minimum upper bound of optimal solutions of those subregions are identified. If the maximal lower bound coincides with the minimal upper bound, an optimal solution has been found; otherwise, the iteration is repeated.

1.3.2. Metaheuristic methods

The metaheuristic method is another type of methods for global optimization. In some literatures, it is also called stochastic method since it more or less use randomly generated numbers or points in its search process. The metaheuristic method originated from 1960s and there were quite a lot of new algorithms have been developed thereafter. Some typical types of metaheuristic methods include the evolutionary algorithms, simulated annealing, tabu search, particle swarming optimization, ant colony algorithm, and so on. In the following, we briefly introduce some of them.

Evolutionary algorithms

The evolutionary algorithm is one of the main types of metaheuristic methods. The genetic algorithm [26, 38, 39, 121], evolution strategy [56, 77] , and evolutionary programming [129, 130] are three typical types of evolutionary algorithms.

The main idea of the genetic algorithm is based upon the biologically natural selection and genetic mechanism. The earliest structure of the genetic algorithm was provided by Glodberg [37]. The genetic algorithm is population-based; it takes into account of a set of candidate solutions instead of only one in each iteration. First, it randomly generates a set of candidate solutions called *the initial population*. One single individual from the population is called a *chromosome*. The number of chromosomes in a population is defined as *population size*. In numerical computation, Chromosomes are coded as binary codes, Gray codes or real-number codes. Then, the offspring is generated in two different ways: *crossover* and *mutation*. Crossover operator randomly exchanges some *genes* (which constitute chromosomes) between two selected individuals. Mutation operator changes some randomly selected genes of an individual in a certain way. After that, a selection pool is constructed by putting the current population and its offspring together. Then, the next population is selected from the selection pool by a certain strategy. In practical computation, a number of maximal generation is set beforehand, this number of maximal generation further plays a role of stopping criterion.

Suppose that $P(t)$ and $O(t)$ represent parents and offspring of the t^{th} generation, respectively. Then, the general structure of genetic algorithm can be described in the following pseudo code.

The General Structure of Genetic Algorithm

```
1 Initialization
    1.1 Generate the initial population $P(0)$,
    1.2 Set crossover rate, mutation rate and maximal generation time,
    1.3 Let $t \leftarrow 0$.
2 While the maximal generation time is not reached, do
    2.1 Crossover and mutation operator: generate $O(t)$,
    2.2 Evaluate $P(t)$ and $O(t)$: compute fitness function,
    2.3 Selection operator: build the next population,
    2.4 $t \leftarrow t + 1$, go to 2.1
    end
end
```

From the pseudo code, we can see that there are three important operators in a genetic algorithm: crossover, mutation and selection operators. Usually different encodings lead to different implementations of operators.

Evolution strategy and evolutionary programming, although proposed for different aims and in different time, are generated from more of less the same idea. The same as the genetic algorithm, they are still population-based method, but evolution strategy and evolutionary programming only use mutation to generate the next generation. The development of evolution strategy went through the following stages: i) two membered evolution strategy, ii) multimembered evolution strategy, iii) $(\mu + \lambda)-$ES and $(\mu, \lambda)-$ES. An excellent survey of evolution strategy refers to [114]. The evolutionary programming is first proposed as an approach to artificial intelligence [86], then applied successfully to many numerical and combinatorial optimization problems [29, 30]. Optimization by evolutionary programming can be summarized into two major steps [130]: i) mutate the solutions in the current population; ii) select the next generation from the mutated and the current solutions.

The Differential evolution

Differential evolution is a metaheuristic method based on the difference of population. It was introduced by Storn and Price [93, 108] in 1996 and 1997 when solving Chebyshew polynomia. The differential evolution is originally developed to solve continuous problems, so it used real-number encoding. The framework of differential evolution is almost the same as the genetic algorithm, mainly including mutation, crossover and selection operators. But the mechanism of those operators are different from those of the genetic algorithm.

Suppose that x_i and x_j are two candidate solutions, then the pair (x_i, x_j) defines a *difference vector* such that

$$d_{i,j} = x_i - x_j.$$

The difference vector is an essential ingredient of mutation mechanism in the differential evolution. The mutation process begins by randomly selecting four population vectors x_1, x_2, x_3 and x_4. Those four vectors are then combined to form the sum of two difference vectors, i.e.,

$$d_{1,2,3,4} = d_{1,2} + d_{3,4} = (x_1 - x_2) + (x_3 - x_4).$$

The difference vectors, as well as the sum of two difference vectors, diminish as population vectors converge. Consequently, difference vectors and the sum of two difference vectors scaled to a size that is appropriate for the population as it evolves. Early versions of differential evolution employed a single difference vector instead of the sum of two difference vectors, but the latter one appeared to facilitate finding an effective scaling factor.

The mutation is operated only on the best solution so far. Suppose that x^* is the best solution so far, and $d_{1,2,3,4}$ is a sum of two difference vectors, then a new candidate solution can be generated by

$$u = x^* + F * d_{1,2,3,4},$$

where F is a *scaling factor* and satisfies $0 < F \leq 1.2$. In fact, this mutation generates some new points which are noisy replica of the current best solution. At the early stage, the noise is heavy since the difference between vectors is large. But as the population converges, the difference vectors become tiny which cause minute destabilization to x^*. Actually, x^* can be replaced by a randomly selected solution, but using x^* speeds convergence. F is determined empirically.

Crossover of the differential evolution is introduced to increase the diversity of perturbed candidate solutions obtained by mutation. Suppose that $x^k = (x_1^k, x_2^k, \cdots, x_n^k)$ is a candidate solution in the kth generation, $u^k = (u_1^k, u_2^k, \cdots, u_n^k)$ is a candidate solution obtained by mutation, then crossover between those two candidate solutions is operated as follows:

$$v_i^{k+1} = \begin{cases} u_i^k & \text{if } \alpha_i \leq \text{CR or } i = \beta \\ x_i^k & \text{if } \alpha_i > \text{CR and } i \neq \beta \end{cases} \qquad i = 1, 2, \cdots, n.$$

Here, α_i is a random number between 0 and 1, CR$\in$ $[0, 1]$ is the crossover constant which plays the role of crossover rate in GA. β is a randomly chosen index among $\{1, 2, \cdots, n\}$ which ensures that v^{k+1} at least gets one component from x^k. Figure 1.2 depicts the mechanism of crossover for $8-$dimensional candidate solutions.

Comparing with the genetic algorithm, the aim of both crossover operators is to increase the diversity of candidate solution. However, their targets are different, in the genetic algorithm the crossover operator operates on two randomly selected current candidate solutions, while in the differential evolution the operator crossover operator operates on one current candidate solution and one mutation candidate solution.

To decide whether or not it should become a member of generation $k + 1$, the crossover candidate solution v^k is compared to the corresponding current candidate solution x^k using the greedy criterion.

$$x^{k+1} = \begin{cases} x^k & \text{if } f(x^k) \leq f(v^k) \\ v^k & \text{if } f(x^k) > f(v^k) \end{cases}$$

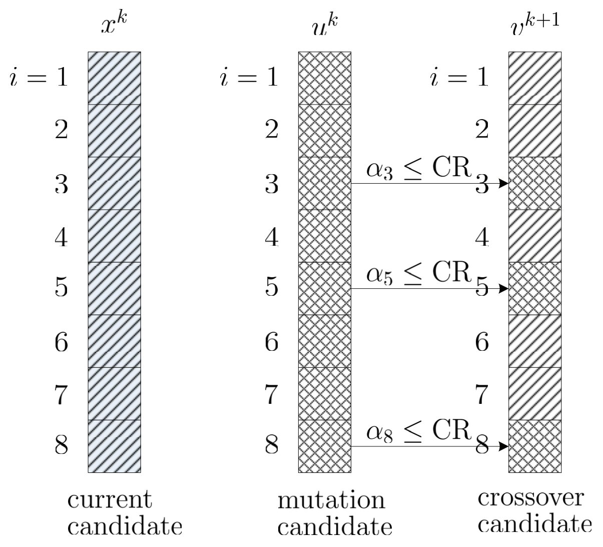


Figure 1.2.: Crossover mechanism for $8-$dimension candidate solutions

The greedy strategy of selection may speed convergence, but it can still cause premature convergence.

The Simulated Annealing

Idea of the simulated annealing method was introduced by N.Metropolis [81]. But it was S.Kirkpatrick [61, 117] and V.Cerny [20] who respectively applied this idea to solve combinatorial optimization and very large scale integration (VLSI) design. The first version of the simulated annealing was applicable only for solving discrete optimization problems. The simulated annealing for solving continuous global optimization problems was developed in late 1980s [16]. C.Zhang and H.Wang [134] applied simulated annealing to solve mixed-integer optimization in 1993.

Motivation of the simulated annealing comes from an analogy between the physical annealing of solids and combinatorial optimization problems. Physical annealing refers to the process of finding low energy states of a solid by initially melting the substance, and then lowering the temperature slowly, spending a long time at temperatures close to the freezing point. An example would be producing a crystal from the molten substance. In a liquid, the particles are arranged randomly. But the ground state of the solid, which corresponds to the minimum energy configuration, will have a particular structure, such as seen in a crystal. If the cooling is not done slowly, the resulting solid will not attain the ground state, but will be frozen into a metastable, locally optimal structure, such as a glass or a crystal with several defects in the structure.

Let a solution x of combinatorial optimization problem equals to a state i of solid; the corresponding objective function value $f(x)$ equals to the energy E_i of state i. Set a parameter t for algorithm which plays the role of temperature T in annealing process. Obviously, the parameter t gradually decrease in algorithm. Then, for every new value of parameter t, the algorithm continuously plays a process of "generate new candidate solution – check the candidate solution – accept/reject". This process actually corresponds to a solid converges a thermal equilibrium in a certain temperature, which can be considered as one implementation of Metropolis algorithm [81]. Metropolis algorithm, starting with an initial state, intend to obtain the final equilibrium state of a system through computing the time evolutionary process. By simulating this process,simulated annealing, starting with an initial candidate solution, can obtain a relatively optimal solution of combinatorial optimization through investigating a great deal of trial solutions. Then, the parameter t is decreased and Metropolis algorithm is implemented again. This process is repeated until the parameter converge to zero, and the approximate global solution of combinatorial optimization can be obtained at last. In the physical annealing process, the temperature must decrease gradually to guarantee the equilibrium state in each temperature, and consequently, converge to the ground state. Therefore, in simulated annealing, the value of parameter t has to decrease gradually as well to guarantee the approximate global optimal solution of combinatorial optimization.

simulated annealing generates a sequence of candidate solutions by Metropolis algorithm. The Metropolis criteria is use to accept or reject a trial solution. In the kth iteration, suppose the controlling parameter is t^k, the current candidate solution is x^k and a generated trial solution x', then the probability of Metropolis algorithm to accept x' is

$$P(x^k \Rightarrow x') = \begin{cases} 1 & \text{if } f(x') \leq f(x^k) \\ \exp\left(\frac{f(x^k)-f(x')}{t^k}\right) & \text{otherwise.} \end{cases} \tag{1.39}$$

Here $t^k \in R^+$ represents the controlling parameter. At the early stage, the value of t is big (which corresponds to the solution temperature of solids), the probability of accepting a uphill trial solution x' is high. This is good for global exploration. On the other hand, after some iterations of transfer, t gradually decrease (which is corresponding to the gradually decrease of solid temperature in annealing), only a very few uphill trial solution can be accepted. This is good for local exploitation. It is one of the key difference of simulated annealing with local search methods that simulated annealing can, in some extent, accept some uphill candidate solutions.

1.3.3. Hybrid methods

The hybrid method is a type of global optimization methods combining both advantages of local search methods and global search methods. Local search methods for convex objective function is descent-oriented, they are efficient and can reach good accuracy of solutions. But these methods only work for local areas or for convex functions, when they are applied to nonconvex functions, they are easily stuck in a local minimizer. Some typical local search methods include the Newton-type method, quasi-Newton method, conjugate gradient method, pattern search method, and so on. On the other hand, global search methods are good at exploring new search areas which have not been visited previously. But it is difficult for the global search methods to obtain solutions with high precision. Typical global search methods include the metaheuristic methods introduced in the previous section. Therefore, it is natural to combine the local exploitation of local search methods and the global exploration of global search methods. Local search methods are applied to precisely search a local minimizer and global search methods are applied to escape the search from a local basin to some other new search areas. In this way, if the objective function has finite number of local basins, the global minimizer can be found by the hybrid method in a high probability.

Hybrid methods are among the most powerful methods in global optimization. There are different approaches to design hybrid methods [6, 7, 42, 132]. Without loss of generality, we can classify these approaches into the following three groups.

1. The first group contains algorithms where global search methods are applied to improve global search properties of local search methods. In [42], a hybrid of the simulated annealing and Nelder-Mead simplex method [83] is developed. A hybrid of the generalized Nelder-Mead method with controlled random search and simulated annealing method is developed in [66]. The paper [92] presents another version of the hybrid of the simulated annealing and simplex methods. A method based on a hybrid of the genetic algorithm and simplex methods is developed in [131]. One can note that all these algorithms use the simplex method as a local search algorithm. This algorithm is efficient when the number of variables is small.

2. The second group contains algorithms where global search methods are applied to escape from a stationary point computed by local search methods and to find a new starting point for local search methods. In the paper [6] a method based on a combination of the discrete gradient and the cutting angle methods is proposed. The discrete gradient method is applied to compute a local minimizer and the cutting angle method is applied to escape from this local minimizer and to find a new starting point for the discrete gradient method. A similar approach is used in the paper [132] to develop a method based on a combination of the simulated annealing and Newton-like methods.

3. The third group contains algorithms where the global method is used to generate a set of initial points for the local search method. Then the local search method is applied starting from each initial point and the best solution is taken as an approximation to

a global solution (see e.g. [75]). Since the local search method is applied repeatedly, algorithms based on such an approach are time-consuming.

One may notice that direct search and derivative-free methods have been more successful than Newton-like methods to develop hybrid methods of global optimization. Results of numerical experiments presented, for example, in [6] show that unlike Newton-like methods, some direct search methods can overcome stationary points which are not local minimizers and even sometimes shallow local minimizers. Therefore, the use of direct search methods allows one to reduce the number of stationary points computed by a local search method and to compute the global minimizer much faster.

Note that the well-known global optimization methods such as the tunneling [73] and filled function methods [124, 125, 126] can also be considered as hybrid methods. These methods also exploit local search method to find local solutions and special mechanisms to escape from those local solutions.

Chapter 2.

A quasisecant method

2.1. Introduction

In this chapter, we consider the following unconstrained optimization problem

$$\text{(NSO)} \quad \begin{cases} \text{Minimize} & f(x) \\ \text{Subject to} & x \in \mathbb{R}^n, \end{cases} \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a locally Lipschitz continuous function and not necessarily differentiable. We call the Problem (2.1) unconstrained nonsmooth optimization problem, abbreviated as NSO.

Numerical methods for solving the Problem (NSO 2.1) have been studied extensively over the last four decades. Given different assumptions on the objective function f, various numerical methods have been presented, such as subgradient methods [107], bundle-type methods [24, 57, 62, 63, 68, 69, 105, 135], the limited memory bundle method [40, 41, 78, 118], gradient sampling methods [18] and methods based on smoothing techniques [27, 84].

For most of the methods listed above, the convexity assumption of the objective function f is needed. For example, in the subgradient method, convexity is necessary for proving convergence; in the bundle-type method, a lower piecewise linear approximation only appears when objective function is convex.

In this chapter, we propose an algorithm for solving Problem (NSO 2.1). The main idea of this algorithm is similar to that of the bundle-type method. However, instead of using a set of subgradients to approximate subdifferential, the new method uses secants or quasisecants. This chapter is structured as follows. First we give definitions of secants and quasisecants, describe an algorithm for finding search directions, then the quasisecant method and finally, we present results of numerical experiments. The quasisecant method was introduced in [2] and all theoretical results given in this chapter are adopted from this paper. However, the implementation of this method is in Matlab and computational results are new and presented first time in this chapter.

2.2. Secants and quasisecants

We first introduce the definitions of secants and quasisecant for locally Lipschitz functions, starting with univariate function. Consider Lipschitz continuous function $\phi : \mathbb{R} \to \mathbb{R}$. A secant is the line passing though at least two points on the graph of the function ϕ. For instance, a line passing through point $(x, \phi(x))$ and $(x+h, \phi(x+h))$, where $h > 0$, is given by

$$l(x) = cx + d,$$

where

$$c = \frac{\phi(x+h) - \phi(x)}{h}, \quad d = \phi(x) - cx.$$

The equation

$$\phi(x+h) - \phi(x) = ch$$

is called the secant equation (see dashed line of Figure 2.1). For univariate smooth function, c becomes derivative when h is infinitely small.

Now, let us consider a locally Lipschitz function $f : \mathbb{R}^n \to \mathbb{R}$. For given $x \in \mathbb{R}^n$ and $g \in S_1 = \{x \in \mathbb{R}^n | \parallel x \parallel = 1\}$, consider function $\phi(t) = f(x + tg), t \in \mathbb{R}, t \geq 0$. Obviously, $\phi(t)$ is a locally Lipschitz continuous function. Therefore, for $h > 0$, we have

$$f(x + hg) - f(x) = \phi(h) - \phi(0) = ch, \tag{2.2}$$

where $c = (f(x - hg) - f(x))/h$. If there exists $u \in \mathbb{R}^n$ such that $c = \langle u, g \rangle$, then we can generalize the definition of secant to function f.

Definition 2.2.1 A vector $u \in \mathbb{R}^n$ is called a *secant* of the function f at the point x in the direction $g \in S_1$ with the length $h > 0$ if

$$f(x + hg) - f(x) = h\langle u, g \rangle.$$

We use the notation $u(x, g, h)$ for any secant of the function f at a point x in the direction $g \in S_1$ with the length $h > 0$. Apparently, the secant closely depends on direction $g \in S_1$ and length $h > 0$. Actually, note the definition of directional derivative (Definition 1.1.1) and the Equation (2.2), we can easily see that $f'(x; g) \approx \langle u, g \rangle$ when h is small. So we can take $\langle u, g \rangle$ as an approximation of directional derivative.

For a given $h > 0$, consider a set-valued mapping $x \mapsto \text{Sec}(x, h)$

$$\text{Sec}(x, h) = \{w \in \mathbb{R}^n | \exists g \in S_1, \text{ such that } w = u(x, g, h)\}.$$

Consider the following set at point x,

$$\text{SL}(x) = \{w = \mathbb{R}^n | \exists g \in S_1 \text{ and } \{x_k\}, h_k \downarrow 0 \text{ as } k \to \infty, \text{ such that } w = \lim_{k \to \infty} u(x, g, h_k)\}.$$

A mapping $x \mapsto \text{Sec}(x, h)$ is called a *subgradient-related (SR)-secant mapping* if the corresponding set $\text{SL}(x) \subset \partial f(x)$ for all $x \in \mathbb{R}^n$.

The computation of secant is not always an easy task. For this reason, we relax the equality to inequality in Equation (2.2) and present a new definition, say quasisecant.

Definition 2.2.2 A vector $v \in \mathbb{R}^n$ is called a *quasisecant* of the function f at the point x in the direction $g \in S_1$ with the length $h > 0$ if

$$f(x + hg) - f(x) \leq h\langle v, g\rangle.$$

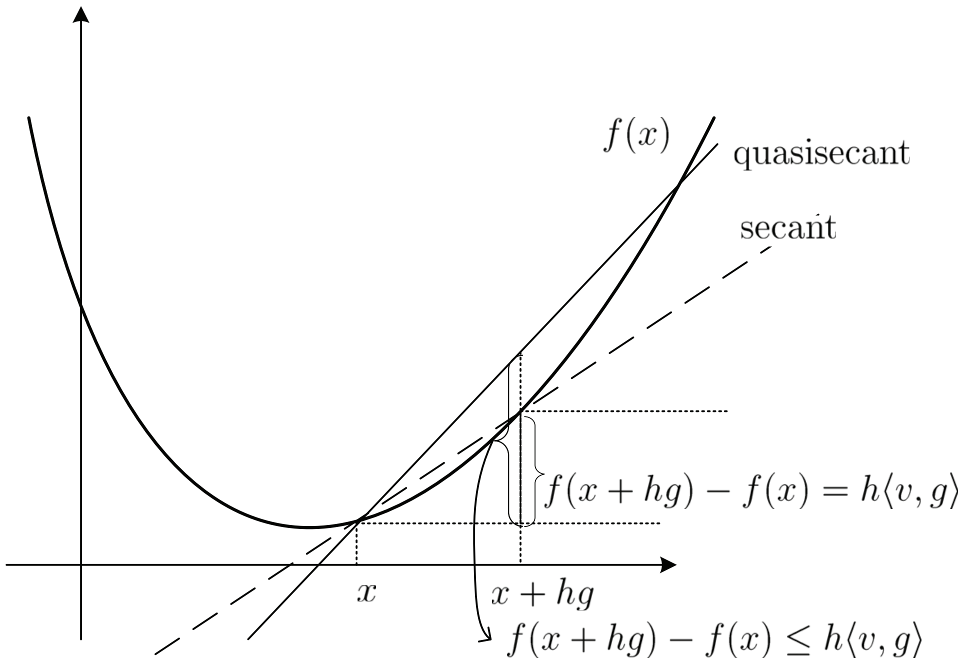
We use the notation $v(x, g, h)$ for any quasisecant of the function f at the point x in the direction $g \in S_1$ with the length $h > 0$ (see Figure 2.1). So for a given $h > 0$, we can define a set-valued mapping

$$\text{QSec}(x, h) = \{w \in \mathbb{R}^n |\ \exists g \in S_1 \text{ such that } w = v(x, g, h)\}.$$

It is clear that any secant is also quasisecant. Therefore, the computation of quasisecants must be easier than the computation of secants. Consider the set of limit points of quasisecant as $h \downarrow 0$,

$$\text{QSL}(x) = \{w = \mathbb{R}^n | \exists g \in S_1 \text{ and } \{x_k\}, h_k \downarrow 0 \text{ as } k \to \infty, \text{ such that } w = \lim_{k \to \infty} v(x, g, h_k)\}.$$

A mapping $x \mapsto \text{Qsec}(x, h)$ is called an *SR-quasisecant mapping* if the corresponding set $\text{QSL}(x) \subset \partial f(x)$ for all $x \in \mathbb{R}^n$. In this case, elements of $\text{Qsec}(x, h)$ are called *SR-quasisecants*. In the quasisecant method, we consider quasisecants instead of secant.



$f(x)$
quasisecant
secant
$f(x+hg)-f(x)=h\langle v,g\rangle$
x
$x+hg$
$f(x+hg)-f(x)\le h\langle v,g\rangle$

Figure 2.1.: Secant and quasisecant for a univariate function

2.3. Computation of a descent direction

We assume that for any bound subset $X \in \mathbb{R}^n$ and any $h_0 > 0$ there exists $K > 0$ such that

$$\| v \| \leq K$$

for all $v \in$ QSec(x,h), $x \in X$ and $h \in (0,h_0]$. Given $x \in \mathbb{R}^n$ and $h > 0$ we consider the following set,

$$W(x,h) = \bar{co}\,\text{QSec}(x,h)$$

where $\bar{co}$ is a closed convex hull of a set. If QSec(x,h) is the SR-quasisecant mapping, then the set $W(x,h)$ is an approximation to the subdifferential $\partial f(x)$. In this case, the set $W(x,h)$ is the convex hull of a set of subgradients computed at points in some neighborhood of x. It is no doubt that the set $W(x,h)$ is compact and convex.

Theorem 2.3.1 Assume that $0_n \notin W(x,h)$, $h > 0$ and

$$\| v^0 \| = \min\{\| v \| \mid v \in W(x,h)\} > 0.$$

Then

$$f(x + hg^0) - f(x) \leq -h \| v^0 \|$$

where $g^0 = - \| v^0 \|^{-1} v^0$.

The proof of Theorem 2.3.1 refers to [2]. Theorem 2.3.1 implies that if $0 \notin W(x,h), h > 0$, then the nearest point of $W(x,h)$, say v^0, can be used to construct a descent direction. However, totally computation of $W(x,h)$ is not always possible. Therefore, we compute only a few elements of $W(x,h)$ and use their convex combination to approximate $W(x,h)$. This process is presented in the following algorithm.

Algorithm 2.3.1 Computation of the descent direction.

step 0: Data: set $h > 0$, $c_1 \in (0,1)$ and a small enough number $\delta > 0$.

step 1: Choose any $g^1 \in S_1$ and compute a quasisecant $v^1 = v(x, g^1, h)$ in the direction g^1. Set $V_1(x) = \{v^1\}$ and $k = 1$.

step 2: Compute $\| \bar{v}^k \|^2 = \min\{\| v \|^2 \mid v \in co\, V_k(x)\}$. If

$$\| \bar{v}^k \| \leq \delta,$$

then stop. Otherwise go to Step 3.

step 3: Compute the search direction by

$$g^{k+1} = -\frac{\bar{v}^k}{\| \bar{v}^k \|}.$$

step 4: If

$$f(x + hg^{k+1}) - f(x) \leq -c_1 h \parallel \bar{v}^k \parallel,$$

then stop the loop, g^{k+1} is a descent direction. Otherwise go to Step 5.

step 5: Compute a quasisecant $v^{k+1} = v(x, g^{k+1}, h)$ in the direction g^{k+1}, construct the set $V_{k+1}(x) = V_k(x) \cup \{v_{k+1}\}$, set $k = k + 1$ and go to Step 2.

It should be note that Algorithm 2.3.1 is based upon, more or less, the same idea as the descent direction finding algorithm in bundle-type methods. The difference is that we use quasisecants in Algorithm 2.3.1 instead of subgradient in bundle-type methods. In Step 2, we calculate the nearest point of a polytope by solving a quadratic problem. Actually,

$$\bar{v}^k = \sum_{i=1}^{k} \lambda_i^* v^i,$$

where $\lambda_i^*, i = 1, 2, \cdots, k$ is the solution of the quadratic problem

$$\begin{cases} \text{Minimize} & \frac{1}{2} \parallel \xi \parallel^2 \\ \text{Subject to} & \xi = \sum_{i=1}^{k} \lambda_i v^i \\ & \sum_{i=1}^{k} \lambda_i = 1 \\ & \lambda_i \geq 0, \ i = 1, 2, \cdots, k. \end{cases} \tag{2.3}$$

This quasisecant programming problem can be solved by the algorithms proposed in [32] or [123]. There are two stop criteria for Algorithm 2.3.1, in Step 2, if $\parallel \bar{v}^k \parallel \leq \delta$, we obtain the so-called $(h, \delta)-$stationary point; in Step 3, if the direction g^{k+1} is a descent direction, we stop the Algorithm 2.3.1 and go to line search. If both of those stop criteria are dissatisfied, in Step 5, we accumulate more information (v_{k+1}) of the neighborhood of the current point and add it to $V_k(x)$. Then, the convex hull of $V_{k+1}(x)$ is taken as a better approximation of $W(x, h)$. Step 5 plays a role of null step like in bundle-type method. Adil [2] proved that, under some normal assumptions, Algorithm 2.3.1 can terminate in finite iterations, say $m > 0$ times. However, in numerical computation, we may not afford m times of iteration. Therefore, we set a smaller positive integer number, say $b \leq m$ to terminate the iteration. This inherits the idea of bundle-type method.

2.4. A quasisecant method

With the help of Algorithm 2.3.1, we can describe a quasisecant method for solving non-smooth optimization problems.

Algorithm 2.4.1 The inner loop

Step 0: Input: the step length $h > 0$, a tolerance parameter $\delta > 0$, parameters $c_1 \in (0,1), c_2 \in (0, c_1]$, the maximal iteration time $b > 0$, the current iteration point x^0 and set $k = 1$. Let $x^k = x^0$

Step 1: Apply Algorithm 2.3.1 on x^k, This algorithm terminates in three possible ways:

option 1 $\| \bar{v}^k \| \leq \delta$, where $\bar{v}^k$ is calculated by solving the quadratic programming problem (2.3);

option 2 the search direction $g^k = - \| \bar{v}^k \|^{-1} \bar{v}^k$ satisfies condition

$$f(x^k + hg^k) - f(x^k) \leq -c_1 h \| \bar{v}^k \|; \tag{2.4}$$

option 3 the maximal iteration time researched.

Step 2: If it is terminated by option 1, we stop the inner loop.

Step 3: If it is terminated by option 2, we go to Step 5 to do the line search.

Step 4: If it is terminated by option 3, we stop the inner loop.

Step 5: Compute $x^{k+1} = x^k + \sigma_k g^k$, where σ_k is defined as follows

$$\sigma_k = \arg\max\{\sigma \geq 0 |\; f(x^k + \sigma g^k) - f(x^k) \leq -c_2 \sigma \parallel \bar{v}^k \parallel\}.$$

Set $k = k + 1$ and go to Step 1.

The following are some explanation of Algorithm 2.4.1. In Step 2, an (h, δ)-stationary point is found. The definiton of (h, δ)-stationary point based on quasisecant and $W(x, h)$. It is a relaxation of classical stationary point in the sense of $v \in W(x, h)$ and $\parallel v \parallel \leq \delta$. In Step 3, Inequality (2.4) stands for the so-called wolf condition for inexact line search. In Step 4, the terminal option 1 and terminal option 2 are not satisfied, but the fixed maximal iteration time is reached, so we have to terminate the inner loop compulsorily. In Step 5, we do the inexact line search, more specifically, we use a double step size strategy in line search. This strategy starts from a small given step size (such as h), then at each iteration, if the current step size decrease objective function value along the considered direction, we accept it and further test its double. Otherwise, we stop line search and take the last accepted step size as a solution.

Algorithm 2.4.1 can be terminated at Step 2 or Step 4. But achieving an (h, δ)-stationary point or reaching the maximal iteration time do not necessarily mean a satisfactory stationary point is obtained. In order to construct a more promising approximation of subdifferential, we have to decrease the value of h. This process is implemented in the following outer loop.

Algorithm 2.4.2 The secant method (The outer loop)

Step 1: Choose any starting point $x^0 \in R^n$, the initial step length $h_0 > 0$, the tolerance parameter $\delta > 0$ and ε, parameter $\alpha \in [0, 1]$. Set $k = 0$.

Step 2: If $0 \in \partial f(x^k)$, then stop.

Step 3: Apply Algorithm 2.4.1 by starting from the point x^k for $h = h_k$ and δ. This algorithm either terminates by obtaining an (h_k, δ)-stationary point or by reaching the maximal iteration time.

Step 4: If $h_k < \varepsilon$, then stop. Otherwise, let $h_{k+1} = \alpha h_k$ and go back to Step 2.

The following are some explanation of Algorithm 2.4.2. Step 2 is the normal necessary condition for nonsmooth optimization problem. This condition is not easy to testify. Like in Step 1 of Algorithm 2.4.1, we can use a small tolerance parameter, i.e., $\| \xi_k \| \leq \varepsilon$, where $\xi_k \in \partial f(x^k)$ and $\varepsilon > 0$ small enough. Another stopping criterion of Algorithm 2.4.2 is $h_k < \varepsilon$ in Step 4. h_k can adjust the approximation of $W(x, h)$, thereby, adjust the approximation of subdifferential, the smaller the value of h_k is the more accurate the approximation can get. However, we should not make h_k very small at the first place, since a small h_k can just approximate objective function locally, whereas a large h_k can approximate objective function in an extensive range, which is needed when solving nonconvex optimization problems.

2.5. Numerical experiments

In this section, we investigate the numerical performance of quasisecant method by testing some benchmarks with nonsmooth objective functions. We consider three types of problems:

(i) Problems with nonsmooth convex objective functions;

(ii) Problems with nonsmooth nonconvex regular objective functions;

(iii) Problems with nonsmooth nonconvex and nonregular objective functions.

Test problems are illustrated in Appendix A. The brief description of test problems are given in the Table 3.1, where the following notations are used:

- n – the number of variables;

- No.$_f$ – the total number of functions under maximum and minimum (if a function contains maximum and minimum functions); and
- f_{opt} – the optimal value.

Table 2.1.: Description of test problems

Function type	Problem	n	No.$_f$	f_{opt}
Nonsmooth convex	Problem 2.1	2	3	1.952245
	Problem 2.2	2	3	0
	Problem 2.3	2	2	0
	Problem 2.4	3	6	3.5997173
	Problem 2.5	4	4	-44
	Problem 2.6	4	4	-44
	Problem 2.7	3	21	0.0042021
	Problem 2.8	4	11	0.0080844
	Problem 2.9	10	2	54.598150
	Problem 2.10	11	10	3.7034827173
Nonsmooth nonconvex regular	Problem 2.11	4	20	115.70644
	Problem 2.12	4	21	0.0026360
	Problem 2.13	4	21	0.0020161
	Problem 2.14	5	21	0.0001224
	Problem 2.15	5	30	0.0223405
	Problem 2.16	6	51	0.0349049
	Problem 2.17	6	41	0.0061853
	Problem 2.18	7	5	680.63006
	Problem 2.19	10	9	24.306209
	Problem 2.20	20	18	93.90525
	Problem 2.21	20	31	0.0000000
	Problem 2.22	11	65	0.047700
Nonsmooth nonconvex nonregular	Problem 2.23	2	6	2
	Problem 2.24	2	-	0
	Problem 2.25	4	-	0

All numerical experiments are run in an environment of MATLAB 2010 installed on an ACER ASPIRE 4730Z laptop with a 2G RAM and a 2.16GB CPU. We use the following two methods for comparison:

- PMIN - a recursive quadratic programming variable metric algorithm for minimax optimization

- PBUN - a proximal bundle algorithm.

For the parameters of the Algorithm 2.4.2, we set $c_1 = 0.2, c_2 = 0.05, \delta = 10^{-7}, h_0 = 1, \alpha = 0.5$ and $\varepsilon = 10^{-7}$.

For the computation of subgradient of maximal function, we take the subgradient of the active function. For example, $f(x) = \max\{f_1(x), f_2(x), \cdots, f_k(x)\}$, where $x \in \mathbb{R}^n$, if $f_{i_0},\ i_0 \in \{1, 2, \cdots, k\}$ is actively at x_0, i.e.,

$$f_{i_0}(x_0) = f(x_0) = \max\{f_1(x_0), f_2(x_0), \cdots, f_k(x_0)\},$$

then, an arbitrary $g \in \partial^T f(x_0)$ is calculated by $g = \nabla f(x_0)$. This strategy for calculation of the subgradient is reasonable, since $f(x)$ is actually differentiable almost everywhere and one subgradient at a differentiable point is the gradient of active function at that point.

First we applied both PMIN and Algorithm 2.4.2 for solving Problems 2.1-2.22 using s-tarting points from [79]. Results are illustrated in Table 2.2. In this table, we present the value of the objective function at the final point (f), the number of function evaluation (n_f) and the number of subgradient evaluation (n_{sub}), respectively. These results demonstrate that, for convex problems (2.1-2.10), both quasisecant method and bundle method successfully solve them with high accuracy. However, for nonconvex problems (2.11-2.22), quasisecant method solves Problems 2.11 to 2.19 with high accuracy, but fails to solve Problems 2.20 and 2.21. This is not out of expectation since quasisecant method is still a local search method. For the number of objective function and subgradient evaluation, bundle method is significantly less than these of the quasisecant method, which implies that there is still some space for improving quasisecant method.

In order to test the stability and statistical properties of the quasisecant method. we run both methods for 100 times using randomly generated starting points. One execution is

Table 2.2.: Results of numercial experiments with given starting points

Problem	Quasisecant method			Bundle method		
	f	n_f	n_{sub}	f	n_f	n_{sub}
Problem 2.1	1.95222565	197	112	1.95222	8	8
Problem 2.2	0	222	138	0	8	8
Problem 2.3	0	1420	686	0	180	94
Problem 2.4	3.59971930	5603	3035	3.59972	15	14
Problem 2.5	-43.999999	3276	1911	-44	16	12
Problem 2.6	-43.283783	182	137	-44	21	13
Problem 2.7	0.00420240	427	237	0.00420	9	9
Problem 2.8	0.00808453	602	347	0.00808	12	11
Problem 2.9	54.5981500	115046	71909	54.59815	88	36
Problem 2.10	3.70348258	890	501	3.70348	18	17
Problem 2.11	115.706509	536	296	115.70644	11	11
Problem 2.12	0.00263605	7451	3960	0.00264	113	36
Problem 2.13	0.00201750	2026	956	0.00202	86	35
Problem 2.14	0.00012410	1072	589	0.00012	8	7
Problem 2.15	0.02234064	772	440	0.02234	57	17
Problem 2.16	0.03490492	706	560	0.03490	53	22
Problem 2.17	0.00619108	1001	777	0.00619	107	20
Problem 2.18	680.630057	71029	45149	680.63006	45	20
Problem 2.19	24.3062090	809	665	24.30621	19	14
Problem 2.20	124.825136	766	694	93.90525	30	21
Problem 2.21	0.52975672	360	353	0.00000	20	19
Problem 2.22	0.04802780	1175	911	0.04803	286	68

called a successful execution if the following condition is satisfied:

$$\frac{f^* - f_{opt}}{1 + |f_{opt}|} \leq \varepsilon,$$

where f^* is the approximate optimal value obtained by the execution, f_{opt} is the best-known optimal value illustrated in Table 3.1, and ε is a tolerance which is set as 10^{-2}. Then, the successful rate (n_s) over 100 simulation is calculated by the following rule:

$$n_s = \frac{\text{number of successful runs}}{100} \times 100\%.$$

The following notation will be used in Table 2.3 and Table 2.4;

- f_{ave} – average of obtained approximate optimal function value;
- f_{std} – standard deviation of obtained approximate optimal function value;
- ave$_f$ – average number of objective function evaluation;
- ave$_g$ – average number of subgradient evaluation;
- ave$_t$ – average number of time consumption.

Table 2.3.: Results of numercial experiments with randomly generated starting points.

Problem	Quasisecant method			Bundle method		
	f_{ave}	f_{std}	n_s	f_{ave}	f_{std}	n_s
Problem 2.1	1.952225	0.000001	100%	1.95222	-	100%
Problem 2.2	1.573017	2.667096	74%	0.90750	-	85%
Problem 2.3	0.000000	0.000000	100%	0	-	100%
Problem 2.4	3.599722	0.000003	100%	3.59972	-	100%
Problem 2.5	-43.999995	0.000007	100%	-44	-	100%
Problem 2.6	-41.237579	1.920560	100%	-28.14011	-	60%
Problem 2.7	0.004202	0.000000	100%	0.03051	-	45%
Problem 2.8	0.028569	0.031030	46%	0.01520	-	10%
Problem 2.9	54.598166	0.000046	100%	54.59815	-	100%
Problem 2.10	3.703483	0.000000	100%	3.70348	-	100%
Problem 2.11	115.706455	0.000016	100%	115.70644	-	100%
Problem 2.12	0.002636	0.000000	100%	0.00264	-	100%
Problem 2.13	0.010519	0.025633	90%	0.02752	-	70%
Problem 2.14	0.544205	0.845076	19%	0.30582	-	15%
Problem 2.15	0.022341	0.000000	100%	0.32527	-	25%
Problem 2.16	0.040736	0.058315	100%	0.00572	-	60%
Problem 2.17	0.013525	0.023933	77%	0.39131	-	10%
Problem 2.18	680.630065	0.000011	100%	680.63006	-	100%
Problem 2.19	24.306209	0.000000	100%	24.30621	-	100%
Problem 2.20	125.763528	2.2081017	0%	93.90525	-	100%
Problem 2.21	0.5297567	0.000000	100%	0.00000	-	100%
Problem 2.22	0.1697883	0.082558	12%	0.22057	-	5%
Problem 2.23	-	-	-	2	-	100%
Problem 2.24	0.000082	0.000049	100%	0.07607	-	85%
Problem 2.25	4.812801	7.552082	16%	2.30008	-	10%

From Table 2.3, for convex problems, the quasisecant method performs better than the bundle method on Problems 2.6 and 2.7 in the sense of better accuracy and higher successful rate. For Problems 2.2 and 2.8, the accuracy obtained by the bundle method is better than that obtained by the quasisecant method, but the successful rate of the quasisecant method on Problem 2.8 is higher than the bundle method. Both of them share the same performance on Problems 2.1, 2.3, 2.4, 2.5, 2.9 and 2.10. For nonconvex problems, the quasisecant method performs better than the bundle method on Problems 2.13-2.17 and 2.22, while worse than the bundle method on Problems 2.20 and 2.21. The numerical performance on Problems 2.11, 2.12, 2.18 and 2.19 is the same for both methods. For nonconvex and nonregular problems, the quasisecant method fails to solve the Problem 2.23, but obtained better performance on the Problem 2.24. On the Problem 2.25, the quasisecant method performs worse on accuracy of the solution but better on successful rate.

From Table 2.4, the quasisecant method needs more objective function and subgradient evaluations than the bundle method for most of the test problems, but there are still some exceptions, such as Problems 2.7, 2.8, 2.12, 2.16, 2.17 and 2.22. Actually, the time consumption of the quasisecant method is not significantly more than that of the bundle method, which implies that the quasisecant method is as efficient as the bundle method.

2.6. Conclusion

In this chapter, we altered the quasisecant method and tested its numerical performance on some academic benchmarks. From the numerical comparison, we can observe that the quasisecant method, although need more objective function and subgradient evaluation, performs as good as the bundle method. Convex nonsmooth problems can be well solved by the quasisecant mehtod. However, it needs to be improved in solving nonconvex nonsmooth and nonconvex nonsmooth nonregular problems.

Table 2.4.: Results of numercial experiments with randomly generated starting points.

Problem	Quasisecant method			Bundle method		
	ave_f	ave_g	ave_t	ave_f	ave_g	ave_t
Problem 2.1	1338	679	0.00	10	10	0.00
Problem 2.2	314	207	0.00	22	9	0.00
Problem 2.3	2091	894	0.00	493	251	0.00
Problem 2.4	859	447	0.00	20	16	0.00
Problem 2.5	782	459	0.00	12	11	0.00
Problem 2.6	423	311	0.00	146	56	0.00
Problem 2.7	221	176	0.00	1626	171	0.01
Problem 2.8	972	500	0.00	57891	4843	0.19
Problem 2.9	2886	1773	0.01	66	35	0.00
Problem 2.10	719689	248592	5.29	65	55	0.00
Problem 2.11	649	376	0.00	29	15	0.00
Problem 2.12	1751	929	0.01	26081	1904	0.12
Problem 2.13	2119	1058	0.01	372	173	0.01
Problem 2.14	1533	885	0.01	61	26	0.00
Problem 2.15	1278	733	0.01	51	23	0.00
Problem 2.16	825	647	0.02	4985	445	0.11
Problem 2.17	2740	2127	0.00	60331	4761	1.10
Problem 2.18	828	483	0.00	58	33	0.00
Problem 2.19	1014	665	0.00	18	15	0.00
Problem 2.20	963	805	0.01	35	26	0.00
Problem 2.21	360	353	0.08	160	52	0.03
Problem 2.22	2033	1533	0.01	66280	4974	2.97
Problem 2.23	-	-	-	32	32	0.00
Problem 2.24	342	172	0.00	22	22	0.00
Problem 2.25	549	370	0.00	37	37	0.00

Chapter 3.

A hybrid quasisecant method for global optimization

3.1. Introduction

In this chapter, we develop a hybrid method for solving the following global optimization problem:

$$\begin{cases} \text{minimize} & f(x) \\ \text{subject to} & x \in [a,b] \end{cases} \tag{3.1}$$

where the objective function f is locally Lipschitz continuous and

$$[a,b] = \{x \in \mathbb{R}^n : \; a_i \leq x_i \leq b_i, \; i = 1, \ldots, n\}.$$

Over the last three decades hybrid methods have become very popular in global optimization. Such methods allows one to take advantages of local and global search methods. Local search methods are fast and more accurate than global search methods. But local search methods are easily trapped in stationary points that are even not local minimizers. On the other hand, global search methods are good at exploring new search areas which have not been visited previously. But global search methods are more time-consuming than local search methods and it is difficult for global search methods to obtain solutions with high accuracy. Hybrid methods combine both the local exploitation of local search methods and global exploration of global search methods. Hybrid methods usually consist of two phases, local search and global search.

Depending on how to combine local and global search methods, we can classify hybrid methods into three groups: i) global search methods are applied to improve global search properties of local search methods [42, 83, 66, 92, 131]; ii)global search methods are applied to escape from a stationary point computed by the local search methods and to find a new starting point for local search methods [6, 132]; iii) global search methods are used to generate a set of starting points for local search methods [75].

One may notice that direct search and derivative-free methods have been more successful than Newton-type methods for developing hybrid methods of global optimization. Results of numerical experiments presented, for example, in [6] show that unlike Newton-type methods, some direct search methods can overcome stationary points which are not local minimizers and even sometimes shallow local minimizers. Therefore, the use of direct search methods allows one to reduce the number of stationary points computed by local search methods and to compute the global minimizer much faster.

Bundle methods are known to be among the most efficient methods in nonsmooth optimization. These methods are based on the use of piecewise linear underestimations of the objective function. The use of (sub)gradients from some neighborhood of the current iteration point brings some elements of global search in these methods. Therefore, this type of methods (including also the cutting plane method which were designed mainly for solving convex problems) are other good candidates to be used as local search methods in hybrid methods of global optimization.

In this chapter, we design a hybrid global optimization method based on the combination of the quasisecant method and a special procedure for escaping local minimizers. More specifically, this procedure is applied to generate starting points from the search space and identify "promising" basins. Then the quasisecant method is applied from points located in these basins to find a set of local minimizers of the objective function. The best local minimizer is accepted as a new approximation to a global minimizer and so on. Numerical results on a number of academic test problems are presented to demonstrate the efficiency of the proposed algorithm. This algorithm is also compared with other hybrid algorithms for global optimization using numerical results.

3.2. The hybrid method

In this section, the quasisecant methods for local search as well as for global search are presented. First, we demonstrate how quasisecants can be used to approximate the objective function in the whole search space.

3.2.1. Approximations using quasisecants

Assume that $v^1, \ldots, v^m, m > 0$ are quasisecants of the function f with the length $h > 0$ computed at a point $x \in \mathbb{R}^n$. Then we can consider the following piecewise linear function:

$$\varphi_m(u) = f(x) + \max_{i=1,\ldots,m} \langle v^i, u - x \rangle.$$

It is clear that φ is an over approximation of f, i.e., $f(u) \leq \varphi_m(u)$ where $\|u - x\| = h$ (See Figure 3.1). Consider the following minimization problem:

$$\begin{cases} \text{minimize} & f(u) \\ \text{subject to} & \|u - x\|^2 = h. \end{cases} \tag{3.2}$$

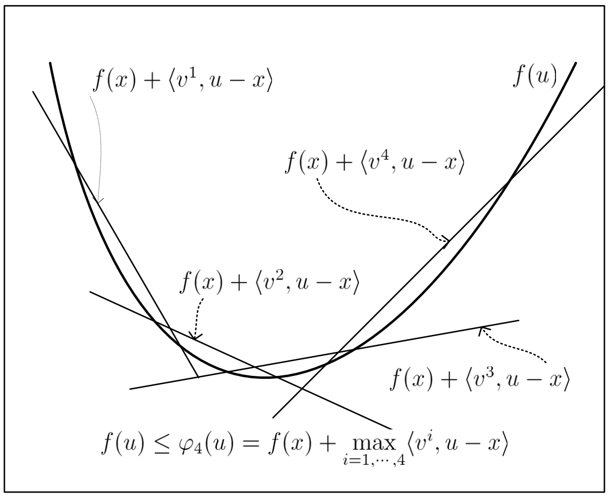

$f(x) + \langle v^1, u - x\rangle$
$f(u)$
$f(x) + \langle v^4, u - x\rangle$
$f(x) + \langle v^2, u - x\rangle$
$f(x) + \langle v^3, u - x\rangle$
$f(u) \le \varphi_4(u) = f(x) + \max_{i=1,\cdots,4} \langle v^i, u - x\rangle$

Figure 3.1.: $\varphi_m(u)$ is some approximation of $f(u)$.

Adding box constraints and solving this problem for each h one can solve the Problem (3.1). However, solving the Problem (3.2) is not an easy task. Therefore, we replace it by the following problem

$$\begin{cases} \text{minimize} & \varphi_m(u) \\ \text{subject to} & \|u - x\|^2 = h. \end{cases} \tag{3.3}$$

Problem (3.3) is a good approximation to the Problem (3.2) if h is sufficiently small, however it is not a good approximation as h increases. However, solving the Problem (3.3) one can find good starting points for solving the Problem (3.2) and consequently for solving the Problem (3.1).

By applying the duality theory, the Problem (3.3) can be reduced to the following problem:

$$\begin{cases} \text{minimize} & \frac{1}{2}\left\|\sum_{i=1}^{m} \alpha_i v^i\right\|^2 \\ \text{subject to} & \\ & \sum_{i=1}^{m} \alpha_i = 1 \\ & \alpha_i \ge 0,\ i = 1, 2, \cdots m. \end{cases}$$

This problem can be solved using quadratic programming algorithms, such as the Wolfe algorithm [122] which is applied in this chapter.

3.2.2. Quasisecant method for local search

Next we give the description of the quasisecant method for local search.

Algorithm 3.2.1 Quasisecant method for local search

Step 0: Select the tolerance $\varepsilon > 0$, sufficiently small number $\lambda_{min} > 0$, numbers $c_1 \in (0,1),\ c_2 \in (0, c_1]$, the starting point $x^0 \in \mathbb{R}^n$ and compute

$$\lambda_{max} = \max\{|x_i^0 - a_i|, |x_i^0 - b_i| :\ i = 1, \dots, n\}.$$

Select a number $\alpha \in (0,1)$, the size of the bundle n_b and set $k := 0$.

Step 1: Compute $\lambda_k = \alpha^k \lambda_{max}$. If $\lambda_k < \lambda_{min}$ then stop. Otherwise set $z := x^k$.

Step 2: Select $d^0 \in S_1$ and set $m := 0$.

Step 3: Compute $y^m = z + \lambda_k d^m$, a subgradient $v^m \in \partial f(y^m)$ and the set

$$Q_m(z) = \left\{v^0, \dots, v^m\right\}.$$

Step 4: Solve the following quadratic programming problem:

$$\text{minimize } \frac{1}{2}\|v\|^2 \text{ subject to } v \in \text{conv } Q_m(z). \tag{3.4}$$

Denote by $\bar{v}$ the solution to this problem.

Step 5: (*Stopping criterion for inner loop*) If

$$\|\bar{v}\| \le \varepsilon \tag{3.5}$$

then go to Step 8.

Step 6: Compute the search direction $d^{m+1} = -\|\bar{v}\|^{-1}\bar{v}$. If

$$f(z + \lambda_k d^{m+1}) - f(z) \leq -c_1\lambda_k\|\bar{v}\| \tag{3.6}$$

then go to Step 7. Otherwise set $m := m + 1$. If $m > n_b$ then go to Step 8. If $m \leq n_b$ then go to Step 3.

Step 7: (*Line search.*) Compute

$$\bar{h} = \arg\max\left\{h > 0 : \; f(z + hd^{m+1}) - f(z) \leq -c_2 h\|\bar{v}\|\right\}.$$

Set $z := z + \bar{h}d^{m+1}$ and go to Step 2.

Step 8: Set $x^{k+1} := z, \; k := k + 1$ and go to Step 1.

Remark 3.2.1 One can see that Algorithm 3.2.1 consists of two loops: outer loop and inner loop. In the outer loop only the parameter λ_k is updated (Step 2). The initial value of this parameter depends on the box constraints. Furthermore, after each outer iteration the point x_k is updated (Step 8). The inner loop consists of Steps 3-7. In this loop one solves the quadratic programming subproblem to find the search directions and carries out line search to find a new point. The parameter λ_k is fixed for all inner loop iterations. The inner loop has two stopping criteria. It stops if the distance between the polytope of quasisecants and the origin is less than a given tolerance or the number of calculated quasisecants reach the maximum number (n_b) allowed.

3.2.3. Quasisecant method for global search

The quasisecant method for global search proceeds as follows.

Algorithm 3.2.2 Quasi-secant method for global search

Step 0: Select the tolerance $\varepsilon > 0$, the base point $x \in \mathbb{R}^n$ and compute

$$\lambda_{max} = \max\{|x_i - a_i|, |x_i - b_i| : \ i = 1, \dots, n\}.$$

Select number $\lambda_{min} \in (0, \lambda_{max})$ and an integer $M > 0$, the size of the bundle n_b. Compute $\beta = (\lambda_{max} - \lambda_{min})/M$. Set $F^0 = \emptyset, \ X^0 = \emptyset$ and $k := 0$.

Step 1: Compute $\lambda_k = \lambda_{min} + k\beta$. If $\lambda_k > \lambda_{max}$ then stop.

Step 2: Select $d^0 \in S_1$ and set $m := 0$.

Step 3: Compute $y^m = x + \lambda_k d^m$ and $f(y^m)$.

Step 4: Compute a subgradient $v^m \in \partial f(y^m)$ and the set

$$Q_m(x) = \left\{v^0, \dots, v^m\right\}.$$

Step 5: Solve the following quadratic programming problem:

$$\text{minimize } \frac{1}{2}\|v\|^2 \text{ subject to } v \in \text{conv } Q_m(x). \tag{3.7}$$

Denote by $\bar{v}$ the solution to this problem.

Step 6: (*Stopping criterion for inner loop*) If

$$\|\bar{v}\| \leq \varepsilon \tag{3.8}$$

then set $X^0 = X^0 \bigcup\{y^m\}, \ F^0 = F^0 \bigcup\{f(y^m)\}, \ k := k + 1$ and go to Step 1. Otherwise compute the search direction $d^{m+1} = -\|\bar{v}\|^{-1}\bar{v}$. Set $m := m + 1$. If

$m > n_b$ then set $X^0 = X^0 \bigcup \{y^m\},\ F^0 = F^0 \bigcup \{f(y^m)\},\ k := k+1$ and go to Step 1, otherwise go to Step 3.

Algorithm 3.2.2 consists of outer and inner loops. In the outer loop the parameter λ_k is updated and in the inner loop the bundle is updated adding one subgradient at each iteration until either the maximum number of subgradients is calculated or the condition (3.5) is met. Then sets X^0 and F^0 are updated by adding the last point generated by the inner loop and the value of the objective function at this point, respectively. The output of Algorithm 3.2.2 is the sets X^0 and F^0.

3.2.4. Hybrid method for global optimization

In order to design a hybrid algorithm for global optimization we need to define the notion of a basin for local minimizers. In many papers the basin is defined as follows (see [35, 115]):

Definition 3.2.1 A basin of f at an isolated minimizer x is a connected domain B which contains x and in which starting from any point the steepest descent trajectory of f converges to x, but outside which the steepest descent trajectory of f does not converge to x.

However, for the purpose of this hybrid algorithm we will use the following definition of the basin. Let x be an isolated local minimizer of f. For each $d \in S_1$ define the following function: $\theta(\alpha) = f(x + \alpha d)$. It is obvious that there exists $\alpha_0 > 0$ such that $\theta(\alpha)$ is an increasing function over the segment $[0, \alpha_0]$. Denote by $\alpha_{max}(d) > 0$ such that this function increasing on the segment $[0, \alpha_{max}(d)]$ and it is not increasing on the segment $[0, \alpha]$ for any $\alpha > \alpha_{max}(d)$.

Definition 3.2.2 A set

$$D(x) = \bigcup_{d \in S_1} D(d),$$

where $D(d) = \{y \in \mathbb{R}^n |\ y = x + \alpha d,\ \alpha \in [0, \alpha_{max}(d)]\}$, is called the basin of the function f at the isolated local minimizer x.

It is obvious that if the set D is the basin in the sense of Definition 3.2.2 then it is also the basin in the sense of Definition 3.2.1. The set

$$B(x) = \{y \in \mathbb{R}^n | \exists\, d \in S_1 \text{ such that } y = x + \alpha_{max}(d)d\}$$

is called the *boundary* of the basin $D(x)$.

Next we describe an algorithm for solving the problem (3.1) which is the hybrid of local and global search quasisecant methods.

Algorithm 3.2.3 Hybrid method for global optimization

Step 0: Select the tolerance $\varepsilon > 0$, the starting point $x^0 \in \mathbb{R}^n$, numbers $\alpha \in (0,1),\ \beta > 1,\ c_1 \in (0,1],\ c_2 \in (0, c_1]$ and the size of bundle n_b. Set $z^1 := x^0,\ x_{best} = x^0,\ f_{best} = f(x^0)$ and $k := 1$.

Step 1: Apply Algorithm 3.2.1 starting from the point z^k using parameters $\alpha,\ c_1,\ c_2, n_b$ to find stationary point x^k of the function f over the set $[a,b]$. If $f(x^k) < f_{best}$ then set $x_{best} := x^k,\ f_{best} := f(x^k)$.

Step 2: Apply Algorithm 3.2.2 using the base point $\bar{x} = x^k$ and the parameters β, n_b. This algorithm generates the set $X^0 = \{y^1, \ldots, y^M\}$ of candidate starting points and the set $F^0 = \{f(y^1), \ldots, f(y^M)\}$ of the values of the function f at these points. Here M is the number of steps in the outer loop of Algorithm 3.2.2.

Step 3: Compute

$$f_{min} = \min\left\{f(y^m) : y^m \in X^0, m = 1, \ldots, M\right\}$$

and

$$\bar{y} = \arg\min\left\{f(y^m) : y^m \in X^0, m = 1, \ldots, M\right\}.$$

If $f_{min} < f_{best}$ then set $z^{k+1} := \bar{y}, k := k + 1$ and go to Step 1. Otherwise go to Step 4.

Step 4: Set $y^0 := x^k$ and find the maximum $m = 0, \dots, M$ with the property: $f(y^l) \geq f(y^{l-1}),\ l = 1, \dots, m$. Update the set X^0 by removing all points $y^1, \dots, y^m$ and the set F^0 by removing values $f(y^1), \dots, f(y^m)$, respectively.

Step 5: Take each point $y \in X^0$ as a starting point and apply Algorithm 3.2.1. As a result a set of stationary points will be generated. Denote by $\bar{x}$ the point with the lowest value of the objective function f among these points. If $f(\bar{x}) < f_{best}$ then set $x^{k+1} = \bar{x}, x_{best} = \bar{x}, f_{best} = f(\bar{x}), k := k + 1$ and go to Step 2. Otherwise the algorithm terminates.

Some explanation on Algorithm 3.2.3 follows. In Step 1 the local search is applied to minimize the objective function. Since the local search algorithm is a descent method in this step the value of the objective function will be improved unless the starting point is already a stationary. This means that if the starting point is not a stationary the new best solution will be found in this step. In Step 2 the global search algorithm is applied to generate starting points in the search space. If any of these points provides lower value than the current best value of the objective function then the point with the lowest value of the objective function is chosen at Step 3. The local search algorithm is applied starting from this point to minimize the objective function. Otherwise in Step 4 all points lying in the basin of the last stationary point are removed from the list of candidate starting points. Then the local search is applied starting from the rest of points to minimize the objective function. The best stationary point among new points is accepted as the next iteration to the global solution in Step 5.

3.2.5. Convergence of Algorithm 3.2.3

Next we prove that inner loops in Algorithms 3.2.1 and 3.2.2 are finite convergent. The inner loop in Algorithm 3.2.1 starts at Step 2 and terminates at Step 5. This loop in Algorithms 3.2.2 consists of Step 2 to Step 6.

Theorem 3.2.1 Assume that there exists a number $G > 0$ such that

$$\|v\| \leq G \qquad \forall v \in \partial f(x) \text{ and } x \in \mathbb{R}^n. \tag{3.9}$$

Inner loops in both Algorithms 3.2.1 and 3.2.2 terminates after finite number of steps and the maximum possible number of such steps is:

$$m = \left\lceil \frac{4}{(1-c_1)^2} \left(\frac{G}{\varepsilon}\right)^4 \right\rceil.$$

Here $\lceil \cdot \rceil$ is a ceiling of a number.

Proof: Notice that for each inner loop λ_k is fixed. We will prove this proposition only for Algorithm 3.2.1. Such proof for Algorithm 3.2.2 is similar. It is obvious that the problem (3.4) is convex. Since $\bar{v}$ is the solution to this problem it follows from the necessary and sufficient condition for a minimum that

$$\langle v, \bar{v} \rangle \geq \|\bar{v}\|^2, \;\; \forall\, v \in \text{conv}\, Q_m(z). \tag{3.10}$$

The inner loop in Algorithm 3.2.1 has two stopping criteria: conditions (3.5) and (3.6). We will prove if the condition (3.6) never happens then the condition (3.5) must happen after finite number of steps.

At each iteration of the inner loop, except the last iteration, none of conditions (3.5) and (3.6) satisfies. This means that $\|\bar{v}\| > \varepsilon$ and

$$f(z + \lambda_k d^{m+1}) - f(z) > -c_1 \lambda_k \|\bar{v}\| \tag{3.11}$$

for $d^{m+1} = -\|\bar{v}\|^{-1}\bar{v}$. Then we prove that the subgradient v^{m+1} computed using the direction d^{m+1} does not belong to the set conv $Q_m(z)$. Since the subgradient v^{m+1} is also quasisecant at the point z we have

$$f(z + \lambda_k d^{m+1}) - f(z) \leq \lambda_k \langle v^{m+1}, d^{m+1} \rangle.$$

Then taking into account that $d^{m+1} = -\bar{v}^{-1}\bar{v}$ and $c_1 \in (0, 1)$, from (3.11) we get

$$\langle v^{m+1}, \bar{v} \rangle < c_1 \|\bar{v}\|^2. \tag{3.12}$$

This inequality and the inequality (3.10) imply that $v^{m+1} \notin \text{conv } Q_m(z)$. This means that at each iteration of the inner loop the approximation of the subdifferential is improved. Next we prove the condition (3.6) must be satisfied after finite many iterations. Denote by $\bar{v}^m$ the solution to the problem (3.4) at the m-th iteration of the inner loop. It is clear that for any $t \in [0, 1]$

$$\|\bar{v}^{m+1}\|^2 \leq \|tv^{m+1} + (1-t)\bar{v}^m\|^2.$$

Then

$$\|\bar{v}^{m+1}\|^2 \leq \|\bar{v}^m\|^2 + 2t\langle \bar{v}^m, v^{m+1} - \bar{v}^m \rangle + t^2 \|v^{m+1} - \bar{v}^m\|^2.$$

Applying (3.12) and the condition (3.9) we get

$$\|\bar{v}^{m+1}\|^2 \leq \|\bar{v}^m\|^2 - 2t(1 - c_1)\|\bar{v}^m\|^2 + 4t^2 G^2. \tag{3.13}$$

Select t as follows:

$$t = \frac{(1 - c_1)\|\bar{v}^m\|^2}{4G^2}.$$

According to the condition (3.9) $t \in (0, 1)$. Putting this t in (3.13) we have

$$\|\bar{v}^{m+1}\|^2 - \|\bar{v}^m\|^2 < -\frac{(1 - c_1)^2 \|\bar{v}^m\|^4}{4G^2}. \tag{3.14}$$

Since $\|\bar{v}^m\| > \varepsilon$ it follows from (3.14) that

$$\|\bar{v}^{m+1}\|^2 - \|\bar{v}^m\|^2 < -\frac{(1-c_1)^2\varepsilon^4}{4G^2}.$$

This inequality is true for all $m = 0, 1, \ldots$. Writing this inequality for $l = 0, \ldots, m$, summing up them and taking into account that $\bar{v}^0 = v^0$ we get

$$\|\bar{v}^{m+1}\|^2 < \|v^0\|^2 - (m+1)\frac{(1-c_1)^2\varepsilon^4}{4G^2}. \tag{3.15}$$

If the inner loop is not finite convergent then by tending $m \to \infty$ from (3.15) we have that $\|\bar{v}^m\| \to -\infty$ which is the contradiction. This proves that the inner loop terminates after finite number of steps. It is obvious that the largest value m for which $\|\bar{v}^{m+1}\| \le \varepsilon$ is

$$m = \left\lceil \frac{4G^2(G^2 - \varepsilon^2)}{(1-c_1)^2\varepsilon^4} \right\rceil.$$

Removing ε from this numerator we get the following estimation for m:

$$m = \left\lceil \frac{4}{(1-c_1)^2}\left(\frac{G}{\varepsilon}\right)^4 \right\rceil.$$

Remark 3.2.2 It follows from the proof of Proposition 3.2.1 that the value of the problem (3.4) decreases sufficiently at each iteration of the inner loop.

Theorem 3.2.2 Assume that the objective function f in Problem (3.1) has finite number of isolated local minimizers. Furthermore, assume that the global search Algorithm 3.2.2 always finds local minimizers over the given sphere. Then Algorithm 3.2.3 finds the global solution to Problem (3.1) after finite number of iterations.

Proof: Since by assumption the global search Algorithm 3.2.2 finds local minimizers over the given sphere at each iteration of Algorithm 3.2.3 the global search algorithm will find new basin never visited so far by Algorithm 3.2.3. This means that at each iteration Algorithm 3.2.3 finds new basin and then applying local search Algorithm 3.2.1 finds a new local minimizer. Since the number of local minimizers is finite, Algorithm 3.2.3 will find the basin of the global minimizer after finite number of steps and this minimizer will reached by the local search algorithm.

3.3. Results of numerical experiments

The efficiency of the proposed algorithm was verified by applying it to some academic test problems and problems from applications with nonconvex objective functions. The algorithm was implemented in Fortran 95 using the gfortran compiler. Numerical experiments were carried out in Intel(R) Core(TM) i5-3470S with CPU 2.90 GHz and RAM 8GB.

Parameters in the proposed algorithm were chosen as follows: $c_1 = 0.2, c_2 = 0.05, \varepsilon = 10^{-7}, n_b = n + 3, \alpha = 0.1$. In the local search $\lambda_{min} = 10^{-8}$ and in the global search $\lambda_{min} = 10^{-4}\lambda_{max}$. We call the proposed algorithm Hybrid Quasisecant Method (HQSM).

3.3.1. Results with academic test problems

In this subsection we present results using 20 academic test problems for global optimization. The brief description of test problems are given in Table 3.1.

We compare the proposed algorithm with the simulated annealing (SA) [76] and the hybrid algorithm SAHPS from [42] using results of numerical experiments. Each algorithm is run $N_r = 100$ times.

The following notation is used in tables of this subsection:

- n - number of variables;
- f_{opt} - optimal value;
- SR - success rate of an algorithm;

- f_{av} - average objective function value over N_r runs of algorithms;
- σ_f - standard deviation of the objective function value over N_r runs of algorithms;
- n_{fav} - average number of objective function evaluations over N_r runs of algorithms;
- σ_{nf} - standard deviation of the number of objective function evaluations over N_r runs of algorithms.

We say that an algorithm terminates successfully if

$$\bar{f} - f_{opt} \leq \varepsilon(|f_{opt}| + 1),$$

where $\bar{f}$ is the best objective function value obtained by an algorithm. In our experiments $\varepsilon = 0.01$. The success rate SR is computed as follows:

$$SR = \frac{S_r}{N_r} \times 100\%,$$

where S_r stands for the number of successful runs.

Table 3.1.: Description of test problems

Problem	n	f_{opt}	Problem	n	f_{opt}
Ackley	$2, 5, 10$	0	Trid	2	-2
Camel	2	-0.0316		5	-30
Griewank	$2, 5, 10$	0		10	-210
Rastrigin	$2, 5, 10$	0	Zakharov	$2, 5, 10$	0
Rosenbrock	$2, 5, 10$	0	Beale	2	0
Schwefel	$2, 5, 10$	0	Bohachvsky1	2	0
Dixon	$2, 5, 10$	0	Bohachvsky2	2	0
Levy	$2, 5, 10$	0	Bohachvsky3	2	0
Michalewicz	2	-1.8013	Branin	2	0.397887
	5	-4.687658	Colville	4	0
	10	-9.66015	Matyas	2	0
Perm	$2, 3$	0	Shubert	2	-186.7309

Results for success rates of algorithms are given in Table 3.2. These results demonstrate that the proposed algorithm can successfully find global solution for most of test problems. Exceptions are Griewank test problem with $n = 5$, Schwefel test problem with $n = 5, 10$, Dixon test problem with $n = 10$ and Michalewicz test problem with $n = 5, 10$. Comparison with other two algorithms show that the proposed algorithm outperforms the SA in all cases and SAHPS algorithm also in all cases except Dixon test problem with $n = 10$. These results also demonstrate that the SAHPS algorithm outperforms the SA in all test problems except Rastrigin problem with $n = 5, 10$ and Levy problem with $n = 10$. The proposed algorithm performs better than other algorithms as the size of problems increase.

Table 3.2.: Success rates by different algorithms (in %)

Problem	n	SA	SAHPS	HQSM	Problem	n	SA	SAHPS	HQSM
	2	3	95	100		2	28	96	100
Ackley	5	1	27	100	Michalewicz	5	0	0	24
	10	1	9	100		10	0	0	0
Camel	2	0	0	100		2	99	100	100
	2	0	2	80	Perm	3	1	42	89
Griewank	5	0	0	42		2	95	100	100
	10	0	0	100	Trid	5	3	100	100
	2	9	100	100		10	0	100	100
Rastrigin	5	10	6	100		2	74	100	100
	10	9	2	100	Zakharov	5	34	100	100
	2	4	100	100		10	12	100	100
Rosenbrock	5	1	81	100	Beale	2	95	93	100
	10	1	85	100	Bohachvsky1	2	0	76	100
	2	0	20	75	Bohachvsky2	2	1	79	100
Schwefel	5	1	3	18	Bohachvsky3	2	1	91	100
	10	0	0	3	Branin	2	37	100	100
	2	58	100	100	Colville	4	24	96	100
Dixon	5	9	56	78	Matyas	2	40	100	100
	10	9	36	2	Shubert	2	7	73	100
	2	26	98	95					
Levy	5	18	42	87					
	10	10	2	86					

Table 3.3 presents the average and standard deviation of best function values obtained by

algorithms in each run. From the table, we can apparently observe that the HQSM outperforms the SA and SAHPS. For those problems where the successful rate is 100%, the average objective function value coincides with the optimal values and the standard deviation of objective function value is small, which implies that the HQSM is robust method. For Griewank problem with $n = 2,\ 5$, Dixon and Michalewicz problems with $n = 5,\ 10$, Levy problem with $n = 2, 5,\ 10$ and Perm problem with $n = 3$ the average objective function is quite close to the optimal value and standard deviation is not large for these problems. This means that the proposed algorithm can find one of the lowest local minimizers where the value of the objective function is close to the optimal value, however the algorithm cannot escape from such minimizers. The proposed algorithm fails to find good solutions for Schwefel problem with $n = 5,\ 10$ where the average objective function value is significantly larger than the optimal objective function value and standard deviations are large. These results also demonstrate that the SA is not very efficient in finding global solutions to these problems since in most cases standard deviations are significantly away from 0 and the SAHPS algorithm is not always robust.

The average number of function evaluations and their standard deviations are given in Table 3.4. Comparing the HQSM with the SA one can see that in most of test problems the former algorithm requires less number of function evaluations than the latter algorithm, except Griewank problem with $n = 2, 5, 10$ and Schwefel, Dixon, Michalewicz problems with $n = 5,\ 10$, Levy problem with $n = 10$. However, one should also take into account that the success rate of the HQSM is much higher than that of the SA in all these problems. The only exception is Dixon problem with $n = 10$. The comparison of the SAHPS and HQSM algorithms shows that the HQSM requires larger number of function evaluations than the SAHPS for most test problems, except Camel and Dixon problems with $n = 2$, Michalewicz problem with $n = 2$, Trid, Zakharov and Matyas problems. Results for success rate and the number of function evaluations show that there is some tradeoff between the number of function evaluations and success rate of algorithms. Despite the fact that the HQSM requires more computational effort than the SAHPS the former is more successful in finding global solutions.

In order to demonstrate the impact of the number M of solved quadratic programming subproblems to find better basins on the performance of the HQSM algorithm we choose two test problems with large number of local solutions: Shubert and Schwefel test problems. Results are presented in Tables 3.5 and 3.6. In these for each value of M we include the success rate, the average number of function and subgradient evaluations and the average CPU time. Graphs of Shubert and Schwefel functions are illustrated in Figures 3.2 and 3.3, respectively.

From Table 3.5, we can obviously observe that when value of M increases, the success rate increases too, which is reasonable according to the design of the HQSM. However, the number of function and subgradient evaluations and CPU time increase as the number M increases which is expected as the number of solved quadratic programming subproblems increase. However, it is easy to see that all these numbers increase linearly depending on M.

Results for Schwefel test problem are different from those for Shubert problem. One can see from Table 3.6 that for $n = 2$ the increase of M improves success rate of the algorithm, however for $m = 5$ and 10 this is not the case. The increase of the M does not lead to any improvement in the performance of the algorithm. This is due to the fact that the proposed model does not always provide good approximation to the objective function. Again we can see that the number of function and subgradient evaluations and CPU time increases linearly depending on M.

3.4. Conclusions

In this chapter a new hybrid method is introduced to solve global optimization problems with box-constraints. An important step in this method is the algorithm to generate starting

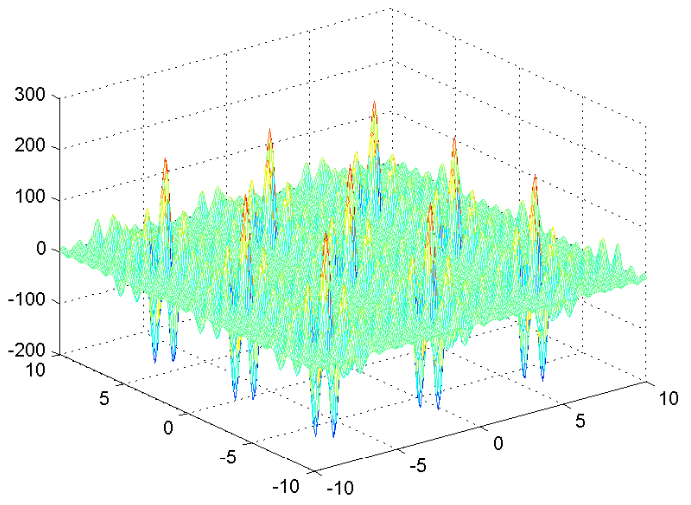


Figure 3.2.: Graph of Shubert function.

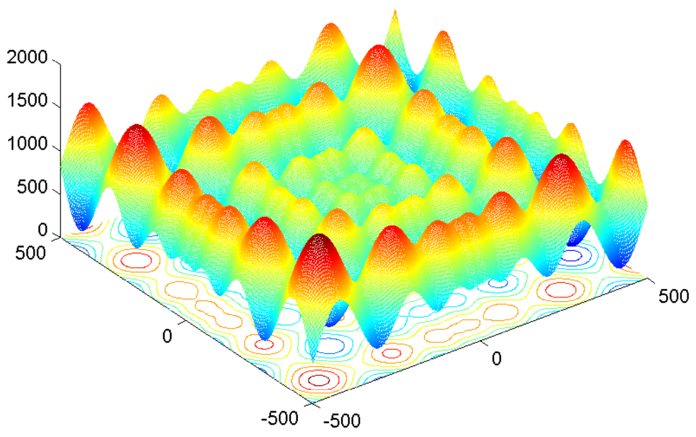


Figure 3.3.: Graph of Schwefel function with 2 variables

points for local search. The main idea is to minimize the objective function over spheres with different radii. In order to do it we replace the objective function by its piecewise linear approximations using quasisecants. The minimization of the piecewise linear functions over sphere with given radius is reduced to a certain quadratic programming problem which can be solved using existing techniques. The quasisecant method of nonsmooth optimization is applied starting from those points to locally minimize the objective function subject to box-constraints. The best solution found is accepted as an approximate global solution. It is proved that the proposed method converges to the global minimizer if the objective function has finite number of local minimizers. Numerical results using 20 global optimization academic test problems and molecular conformation problem demonstrate that the proposed method is efficient for solving wide range of global optimization problems. Comparison of the proposed method with the simulated annealing and one hybrid method of global optimization shows that the proposed method outperforms other two methods.

Table 3.3.: Average function values and standard deviations

Problem	n	SA		SAHPS		HQSM	
		f_{av}	σ_f	f_{av}	σ_f	f_{av}	σ_f
Ackley	2	16.0672	5.0856	0.1801	0.8381	0.0000	0.0000
	5	16.7302	4.2031	7.4112	6.8959	0.0000	0.0000
	10	16.5019	4.0660	14.6924	5.5004	0.0000	0.0000
Camel	2	39.1564	130.9001	0.0000	0.0000	-1.0316	0.0000
Griewank	2	57.6895	54.2706	1.1158	2.8769	0.0123	0.0123
	5	187.5046	141.9986	2.1006	1.4176	0.0176	0.0148
	10	326.4451	281.3132	1.3392	1.1203	0.0000	0.0000
Rastrigin	2	20.1179	17.1816	0.0000	0.0000	0.0000	0.0000
	5	47.0612	41.0121	11.3524	7.9597	0.0000	0.0000
	10	88.1926	71.4284	37.8182	15.7585	0.0000	0.0000
Rosenbrock	2	29.3877	94.7172	0.0000	0.0000	0.0000	0.0000
	5	37269.7502	122892	0.6324	1.4470	0.0000	0.0000
	10	564330	1046976	0.7574	1.5718	0.0000	0.0000
Schwefel	2	816.9584	432.0649	161.7460	119.5814	36.7159	68.7832
	5	2141.6841	982.1579	594.1933	224.6391	245.1676	161.3249
	10	3770.5268	2073.3792	1485.7170	349.9941	631.1999	255.6704
Dixon	2	53.0085	168.9965	0.0000	0.0000	0.0000	0.0000
	5	25163.1046	41661	0.2868	0.3315	0.1466	0.2775
	10	183710	266653	0.4266	0.3216	0.6533	0.0938
Levy	2	4.4282	6.5635	0.0690	0.4862	0.0580	0.2541
	5	9.4886	11.2778	1.3248	1.7457	0.1546	0.5464
	10	25.9554	24.9287	6.5857	4.4839	0.2717	0.8305
Michalewicz	2	-1.0458	0.6635	-1.7736	0.1406	-1.8013	0.0000
	5	-0.8859	0.7185	-1.7542	0.1600	-4.4236	0.2680
	10	-0.7319	0.5851	-1.7206	0.2114	-8.3856	0.6143
Perm	2	0.0002	0.0023	0.0000	0.0000	0.0000	0.0000
	3	66.5102	184.5437	0.0072	0.0049	0.0020	0.0038
Trid	2	-1.9847	0.0715	-1.9999	0.0000	-2.0000	0.0000
	5	146.5503	207.6424	-29.9999	0.0000	-30.0000	0.0000
	10	3254.4697	3126.3721	-209.9999	0.0000	-209.9999	0.0000
Zakharov	2	50.2058	154.7960	0.0000	0.0000	0.0000	0.0000
	5	1199630	2147537	0.0000	0.0000	0.0000	0.0000
	10	469841326	747566805	0.0000	0.0000	0.0000	0.0000
Beale	2	0.0159	0.0899	0.0533	0.1954	0.0000	0.0000
Bohachevsky1	2	7928.8528	8181.2766	0.1113	0.2089	0.0000	0.0000
Bohachevsky2	2	8365.6463	7836.6911	0.0458	0.0893	0.0000	0.0000
Bohachevsky3	2	7993.9742	8120.3851	0.0203	0.0650	0.0000	0.0000
Brain	2	2.5801	2.9185	0.3978	0.0000	0.3978	0.0000
Colville	4	41344.1868	98024	0.1351	0.6954	0.0000	0.0000
Matyas	2	0.2979	0.3783	0.0000	0.0000	0.0000	0.0000
Shubert	2	-41.0453	53.4556	-166.6263	35.5949	-186.7309	0.0000

Table 3.4.: The number of function evaluations and standard deviations

Problem	n	SA		SAHPS		HQSM	
		nf_{av}	σ_{nf}	nf_{av}	σ_{nf}	nf_{av}	σ_{nf}
Ackley	2	7863	689	305	62	4741	1538
	5	7319	599	638	180	5453	1035
	10	6209	1159	1137	382	6435	1276
Camel	2	4755	1109	216	27	199	47
Griewank	2	3907	388	260	39	29336	4806
	5	2767	413	512	66	53701	4158
	10	2560	87	1002	0	19084	5673
Rastrigin	2	9737	805	297	45	670	226
	5	10156	814	584	147	1087	714
	10	10658	1101	1066	262	1558	961
Rosenbrock	2	11353	3967	237	50	744	37
	5	6295	4590	502	5	1670	140
	10	5203	3703	1004	29	2329	423
Schwefel	2	2900	369	304	54	3840	704
	5	2868	152	582	146	19251	3943
	10	2852	119	1060	245	11254	2140
Dixon	2	4834	1423	221	37	182	22
	5	3749	1218	502	6	6541	400
	10	3724	1459	1002	0	11301	1537
Levy	2	5708	1660	256	46	2515	1447
	5	5577	1802	520	62	4441	2520
	10	4740	1979	1008	44	11440	6165
Michalewics	2	4451	2594	290	65	205	62
	5	1885	237	584	169	3202	961
	10	1704	72	1014	94	28756	7753
Perm	2	7161	2216	211	22	218	22
	3	7539	4329	302	4	1546	587
Trid	2	4288	771	203	3	137	16
	5	2961	226	502	0	260	22
	10	2830	48	1002	0	433	62
Zakharov	2	4673	1100	219	34	130	16
	5	4021	1284	502	3	191	32
	10	3813	1791	1002	0	306	32
Beale	2	5462	1044	238	43	655	140
Bohachevsky1	2	3415	1173	220	18	1517	70
Bohachevsky2	2	3282	790	227	29	2914	65
Bohachevsky3	2	3482	1140	228	32	2974	107
Brain	2	3908	750	260	43	279	30
Colville	4	5328	3546	405	15	961	90
Matyas	2	2944	455	239	41	159	40
Shubert	2	10253	1218	312	48	10415	802

Table 3.5.: Results of Shubert problem with different value of M.

n	M	SR	nf_{av}	ng_{av}	t_{av}
2	10	42	363	266	0.002
	50	78	1484	1117	0.007
	100	89	2311	1819	0.010
	200	96	5084	3902	0.024
	300	99	6785	5326	0.031
	500	100	10415	8397	0.051

Table 3.6.: Results of Schwefel problem with different values of M.

n	M	SR	nf_{av}	ng_{av}	t_{av}
2	10	19	428	276	0.002
	50	67	1544	1074	0.007
	100	76	2496	1813	0.012
	200	75	3840	3027	0.022
5	10	3	600	449	0.008
	50	13	2822	2203	0.053
	100	8	5260	4251	0.104
	200	22	9041	7754	0.249
10	10	0	626	488	0.004
	50	1	3298	2650	0.056
	100	4	6308	5293	0.185
	200	3	11254	9775	0.417

Chapter 4.

A hybrid method for constrained global optimization

4.1. Introduction

In the previous chapter, we discussed the global optimization problems with box constraints. However, most global optimization problems appearing in the practical and engineering applications are with constraints. For example, a company intends to maximize its profit, but the resources and investment are limited; a restaurant wants to minimize its cost, but the customers' requirements have to be satisfied, and so on. The optimization problems which model these problems are called constrained optimization problems. A general formulation of constrained optimization problems is as follows:

$$\begin{cases} \text{Minimize} & f(x) \\ \text{Subject to} & g_i(x) \le 0, \quad i = 1, 2, \cdots, l \\ & h_j(x) = 0 \quad j = 1, 2, \cdots, m \\ & x \in X, \end{cases} \tag{4.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*, $g_i : \mathbb{R}^n \to \mathbb{R},\ i = 1, 2, \cdots, l$ are *inequality constraint functions*, $h_j : \mathbb{R}^n \to \mathbb{R},\ j = 1, 2, \cdots, m$ are *equality constraint functions*, and

$$X = \{x = (x_1, x_2, \cdots, x_n)|\ l_i \leq x_i \leq u_i \quad i = 1, 2, \cdots, n\}$$

is a box set, $l = (l_1, l_2, \cdots, l_n)^T$ and $u = (u_1, u_2, \cdots, u_n)^T$ are the *lower bound* and *upper bound*, respectively.

Definition 4.1.1 The *search space* of the Problem (4.1) is defined as

$$S = \{x \in \mathbb{R}^n | x \in X\}.$$

Furthermore, the *feasible search space* is defined as

$$F = \{x \in S | g_i(x) \leq 0,\ i = 1, 2, \cdots, l \quad h_j(x) = 0,\ j = 1, 2, \cdots, m\}.$$

Any point $x \in F$ is called a *feasible point*.

Definition 4.1.2 A point $x^* \in \mathbb{R}^n$ is called a *local minimizer* of the Problem (4.1), if

i) x^* is a feasible point, i.e., $x \in F$,

ii) there exist a feasible $\varepsilon-$neighborhood of x^*, say $B = B(x^*, \varepsilon) \bigcap F$, such that, for any $x \in B$, we have

$$f(x^*) \leq f(x).$$

The objective function value $f^* = f(x^*)$ is called a *local minimum* of the Problem (4.1).

Definition 4.1.3 A point $x^* \in \mathbb{R}^n$ is called a *global minimizer* of the Problem (4.1), if

i) x^* is a feasible point, i.e., $x \in F$,

ii) for any $x \in F$, we have

$$f(x^*) \leq f(x).$$

The objective function value $f^* = f(x^*)$ is called the *global minimum* of the Problem (4.1).

Obviously, the local/global minimizer of a constrained optimization problem may be different from the same problem without constraints, because the local/global minimizer for the problem without constraints may be excluded by the constraints in the problem with constraints.

Methods for solving constrained optimization problems include auxiliary function methods and direct methods. Auxiliary function methods (which are still called indirect methods) transform constrained optimization problems into unconstrained or box-constrained optimization problems by applying auxiliary functions. Two typical auxiliary functions are the *penalty function* and *barrier function method.* The core idea of penalty function method is to replace the inequality and equality constraints by appending the violations of constraints into the objective function. In other words, the auxiliary function is constructed by adding a scalar of the violation of constraints to the objective function. Then, the original constrained optimization problem is solved by minimizing the auxiliary function over a box constraints.

The numerical performance of the penalty function method closely depends on the penalty parameter. However, choosing a suitable parameter is a difficult issue for the penalty function method. On the one hand, a small penalty parameter makes the auxiliary function easy to be minimized but cannot exclude all the infeasible points. On the other hand, a large penalty parameter makes the global minimizer of the auxiliary function to be closer to the real global minimizer of the original constrained optimization problem, but it may cause the so-called ill-conditioning of the auxiliary function.

Similar to the penalty functions, the barrier functions are also used to transform a constrained problem into an unconstrained problem or a sequence of unconstrained problems. These functions set a barrier against leaving the feasible region. If the optimal solution occurs at the boundary of the feasible region, the procedure moves from the interior to the boundary. The barrier function method can only apply for the optimization problems with inequality constraints and box constraints. The barrier function is nonnegative and continuous over the region $\{x \in \mathbb{R}^n | g_i(x) < 0,\ i = 1, 2, \cdots, l\}$ and approaches ∞ as the boundary of the region $\{x \in \mathbb{R}^n | g_i(x) \leq 0,\ i = 1, 2, \cdots, l\}$ is approached from the interior.

One drawback of the barrier function method is that the search process has to be started from the inside of the feasible search space. But for some constrained optimization problems, finding a feasible point is a not an easy task.

Genetic and evolutionary algorithms are among direct methods for constrained global optimization. There are two approaches to handle the constraints when one applies metaheuristics to solve constrained optimization problems:, i) absolutely exclude infeasible points, which is still called the death penalty method, ii) methods based on the multi-objective optimization concepts. The first approach is simple and easy to implement, but without good efficiency, because for some constrained optimization problems, it is very difficult to obtain a feasible point, let alone to exclude it. The second approach transforms a constrained optimization problem into a multi-objective optimization problem with two objectives. One is the original objective function, the another one is the feasibility violation function. In this way, solving the original constrained optimization problem is equivalent to search a special Pareto solution (the first objective value of this solution is the optimal value of the original constrained optimization problem, the second objective value is the feasibility violation at this solution which is less or equal to zero) of the multi-objective optimization problem.

In this chapter, we design a hybrid method for constrained global optimization. This method combines the genetic algorithm and Hooke Jeeves method. The constraints are handled by the penalty function method. We investigate two penalty functions: the quadratic penalty function and exact penalty function. A hybrid method is developed to solve the auxiliary functions constructed using penalty functions.

4.2. Constraints handling technique by penalty function method

The penalty function method transforms a constrained optimization problem into a sequence of unconstrained optimization problems. The constraints are appended to the objective function via a penalty parameter and a penalty function. In general, a penalty function admits a positive penalty for infeasible points and no penalty for feasible points. Taking the Problem (4.1) for example, for the inequality constraints $g_i(x) \leq 0,\ i = 1, \ldots, l$ and equality constraints $h_j(x) = 0,\ j = 1, \ldots, m$, a feasible penalty function is in the form of

$$\alpha(x) = \sum_{i=1}^{l} \phi[g_i(x)] + \sum_{j=1}^{m} \psi[h_j(x)], \tag{4.2}$$

where ϕ and ψ are continuous functions satisfying the following conditions,

$$\phi(y) = 0 \quad \text{if } y \leq 0 \quad \text{and} \quad \phi(y) > 0 \quad \text{if } y > 0;$$

$$\psi(y) = 0 \quad \text{if } y = 0 \quad \text{and} \quad \psi(y) > 0 \quad \text{if } y \neq 0.$$

More specifically, ϕ and ψ are of the form

$$\begin{aligned} \phi(y) &= [\max\{0, y\}]^p, \\ \psi(y) &= |y|^p, \end{aligned}$$

where p is a positive integer. For such case, the penalty function α can be written as

$$\alpha_p(x) = \sum_{i=1}^{m} [\max\{0, g_i(x)\}]^p + \sum_{j=1}^{l} |h_j(x)|^p. \tag{4.3}$$

Let $g = (g_1, g_2, \ldots, g_l)^T$ and $h = (h_1, h_2, \ldots, h_m)^T$ be vector functions on $\mathbb{R}^n$ and $X = [l, u]$ be a box set. Then, the Problem (4.1) can be rewritten in a vector form which is considered as the *primal problem*,

$$\begin{cases} \text{Minimize} & f(x) \\ \text{Subject to} & g(x) \le 0 \\ & h(x) = 0 \\ & x \in X. \end{cases} \tag{4.4}$$

Using the the penalty function supplied in (4.2), a *Penalty Problem* corresponding to (4.4) can be stated as

$$\begin{cases} \text{Maximize} & \theta(\mu) \\ \text{Subject to} & \mu \ge 0, \end{cases}$$

where $\theta(\mu) = \inf\{f(x) + \mu\alpha(x) |\ x \in X\}$ and μ is the penalty parameter. An important relationship between the primal problem and penalty problem is [9]

$$\inf\{f(x) |\ x \in X,\ g(x) \le 0,\ h(x) = 0\} = \sup_{\mu \ge 0} \theta(\mu) = \lim_{\mu \to \infty} \theta(\mu).$$

From this relationship, it is clear that we can get arbitrarily close to the optimal objective function value of the primal problem by solving $\theta(\mu)$ for a sufficiently large μ.

Set $p = 2$ in (4.3), we obtain a quadratic penalty function

$$\alpha_2(x) = \sum_{i=1}^{m} [\max\{0,\ g_i(x)\}]^2 + \sum_{j=1}^{l} |h_j(x)|^2,$$

and the corresponding *auxiliary problem* is

$$\text{(Model 1):} \quad \begin{cases} \text{Minimize} & f(x) + \mu\alpha_2(x) \\ \text{Subject to} & x \in X. \end{cases} \tag{4.5}$$

For each penalty parameter μ_k, let x_k^* be an optimal solution of (4.5). Then, x_k^* can be considered as an approximate solution of the primal problem. From the relationship between the primal problem and the penalty problem, we know that $x_k^* \to x^*$ when $\mu_k \to \infty$, where x^* is a solution of the primal problem. Thus, the better approximation the solution can achieve, the larger penalty parameter μ_k is required. However, a large parameter μ_k makes (4.5) encounter the so-called ill-condition which may cause serious computational difficulties [9].

From the numerical point of view, initial point plays a key role in solving the Problem (4.5), especially when the penalty parameter μ is large. Most algorithms use the penalty functions with a sequence of increasing penalty parameters. During the penalty parameter updating, the approximate optimal solution of the Problem (4.5) with the old parameter is taken as an initial point for solving the Problem (4.5) with the new parameter.

To avoid the penalty parameter μ to be arbitrary large, exact penalty function method was developed. In (4.3), let $p = 1$, then

$$\alpha_1(x) = \sum_{i=1}^{m}[\max\{0, g_i(x)\}] + \sum_{j=1}^{l} |h_j(x)|$$

and the corresponding *auxiliary problem* is

$$\text{(Model 2):} \quad \begin{cases} \text{Minimize} \quad f(x) + \mu\alpha_1(x) \\ \text{Subject to} \quad x \in X. \end{cases} . \tag{4.6}$$

It can be proved that under certain regular assumptions, when μ exceeds a threshold, the solution of the auxiliary problem (4.6) is exactly as the solution of the primal problem [133]. The advantage of the exact penalty function is that the penalty parameter does not need to be infinite. However, the Problem (4.6) is nonsmooth because of the existence of maximum and absolute functions. Thus, the gradient-based methods may not be fit for solving the Problem (4.6).

4.3. A hybrid method for constrained optimization

In this section, we develop a new hybrid method for solving the Problem (4.5) and (4.6). This method combines the genetic algorithm [26, 38, 39, 121] and Hooke Jeeves method [44]. An acceleration operator is embedded into the general procedure of genetic algorithm. This acceleration operator is based upon the the Hooke Jeeves method. In each generation of the genetic algorithm, the acceleration operator choose some individuals as the starting points for local search. The acceleration operator can improve the convergence rate and accuracy of the genetic algorithm. Because the genetic algorithm and Hooke Jeeves method are both derivative-free methods, the hybrid method is derivative-free. This property of the hybrid method can overcome the computational difficulty caused by large penalty parameter of the Problem (4.5) and nonsmoothness of the Problem (4.5).

4.3.1. The genetic algorithm

The main idea of the genetic algorithm is based upon the biologically natural selection and genetic mechanism. The earliest structure of genetic algorithm was provided by Glodberg [37]. It first randomly generates a series of solutions which is called the initial population, and one single individual from the population is called a chromosome. The number of chromosomes in a population is defined as the population size. In numerical computation, Chromosomes are expressed as binary code, Gray code or real-number code. Those chromosomes generate their offspring in two different ways: crossover and mutation. Crossover randomly exchanges some genes (which constitute chromosomes) between two selected individuals. Mutation changes some randomly selected genes of an individual in a certain way. Then a selection pool is constructed by putting the offspring generated from crossover and mutation and the next population is selected from the selection pool. The criterion for selecting the next generation is the performance of each chromosome according to a fitness function which is normally the objection function. These chromosomes whose fitness value are smaller are kept and whose fitness are larger are eliminated. In this way, as the generation iteration goes on, the algorithm will converge to the best chromosome, which probably is the optimal solution or suboptimal solution of the original optimization problem. In practical computation, we set beforehand a maximal generation time, this maximal generation time plays a role of stopping criterion.

Suppose that $P(t)$ and $O(t)$ represent parents and offsprings of the t^{th} generation, respectively. Then, the general structure of genetic algorithm can be described in the following pseudo code.

General Structure of Genetic Algorithm

```
1 Initialization
    1.1 Generate the initial population $P(0)$,
    1.2 Set crossover rate, mutation rate and maximal generation time,
    1.3 Let $t \leftarrow 0$.
2 While the maximal generation time is not reached, do
    2.1 Crossover operator: generate $O_c(t)$,
    2.2 Mutation operator: generate $O_m(t)$,
    2.3 Construct the selection pool by doing $O(t) = O_c(t) \cup O_m(t)$,
    2.3 Evaluate$O(t)$: compute the value of fitness function for each offspring,
    2.4 Selection operator: select the next generation,
    2.5 $t \leftarrow t + 1$, go to 2.1
      end
end
```

From the pseudo code, we can see that there are three important operators in a general genetic algorithm: crossover operator, mutation operator and selection operator. Usually different encodings leads to different implementations of operators. In this method, we use the arithmetic crossover operator, nonuniform mutation operator. There are briefly described as follows.

Initial population generator

Note that $X = [l, u]$ is the box constraint in the Problem (4.4), where $l, u \in \mathbb{R}^n$ are vectors. During the implementation of genetic algorithm, the initial population is randomly generated from X, and the number of chromosomes in the initial population equals population size. The process for generating initial population is illustrated as follows.

Step 1: Input population size: $popu\_size$, upper bound u and lower bound l, respectively. Set $k = 1$.

Step 2: if $k \leq popu\_size$, then generate the k^{th} initial chromosome by

$$x_i = l_i + \alpha_i(u_i - l_i), \quad i = 1, 2, \dots, n,$$

where $\alpha_i,\ i = 1, 2, \dots, n$ are randomly chosen number in $[0, 1]$.

Step 3: Set $k = k + 1$ and go back to Step2.

Arithmetic crossover operator

For the crossover operator, we use arithmetic crossover. Suppose that x_1 and x_2 are two chromosomes randomly selected to crossover, then the following rule is used to generate their offsprings

$$\begin{aligned} x_1' &= \beta x_1 + (1 - \beta)x_2 \\ x_2' &= \beta x_2 + (1 - \beta)x_1, \end{aligned} \tag{4.7}$$

where $\beta \in [0, 1]$ is a random number. The process of arithmetic crossover operator is given as follows.

Step 1: Input crossover rate: $cross\_rate$, and population size: $popu\_size$. Let counter $k = 1$.

Step 2: When $k \leq popu\_size$, generate a number $\alpha \in [0, 1]$, if $\alpha \leq cross\_rate$, then the k^{th} chromosome is marked as a candidate to crossover; otherwise, set $k = k + 1$ and go back to Step 2

Step 3: Sequently choose two chromosomes which were marked as candidates for crossover, and crossover them using the strategy (4.7). The chromosomes obtained are stored as offspring.

Chromosomes generated by arithmetic crossover are actually convex combinations of x_1 and x_2. This crossover operator, on the one hand, sufficiently searches the local area, on the other hand, guarantees some global exploration. Additionally, this strategy is simple, direct and easy to implement. The drawback of this crossover is that only those points between x_1 and x_2 are considered, which reduces search area of crossover operator. To overcome it, we enlarge the random number β from $[0, 1]$ to $[-1, 1]$.

Nonuniform mutation operator

Nonuniform mutation is applied in mutation operator. For a given parent x, if its component x_k (here, subscript represents the kth element of vector x) was chosen to mutate, then the offspring should be

$$x' = (x_1, \ldots, x'_k, \ldots, x_n).$$

Here, x'_k is randomly chosen from the following two options

$$\begin{aligned} & x_k?x_k + d(t, x_k^U - x_k) \qquad \text{if } \gamma \le 0.5 \\ \text{or} & \\ & x'_k = x_k + d(t, x_k - x_k^L) \quad \text{if } \gamma > 0.5, \end{aligned} \tag{4.8}$$

where $\gamma \in [0, 1]$ is a randomly chosen number, x_k^U and x_k^L are upper bound and lower bound of x_k, respectively. The function $d(t, y)$ is chosen to satisfy: $d(t, y) \in [0, y]$ and $\lim_{t\to\infty} d(t, y) = 0$. For example,

$$d(t, y) = yr(1 - \frac{t}{T})^b,$$

where r is a random number between 0 and 1, T is the maximal generation time, and b is the parameter for nonuniform degree which is set to be 1 in our numerical tests. The function $d(t, y)$ allows a large mutation of the selected chromosome at the earlier generations but a slight mutation when iteration achieves the maximum generation time. This trend is reasonable since at the early generations global exploration is emphasized and at the later generations local exploitation is emphasized. The process of nonuniform mutation operator is described as follows.

Step 1: Input the mutation rate: $mutate\_rate$, population size: $popu\_size$, dimension of the problem: n, upper bound u, lower bound l and the maximal generation time: $T = max\_gene$. Set counter $i = 1$ for chromosomes, set counter $k = 1$ for elements of each chromosome.

Step 2: For the ith chromosome (denoted as x), when $k \le n$, generate a number $\alpha \in [0, 1]$, if $\alpha \le mutate\_rate$, then the element x_k mutates according to the strategies provided in (4.8); otherwise, set $k = k + 1$ and go back to Step 2.

Step 3 If $i < popu\_size$, then, let $i = i + 1$ and $k = 1$, go back to Step 2; otherwise, stop

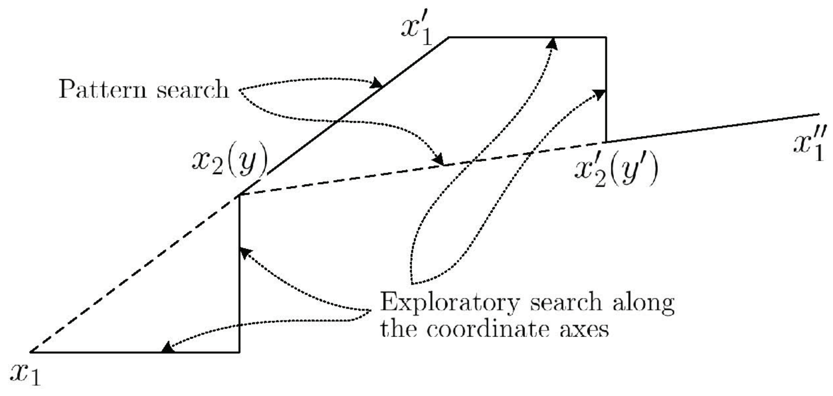


Figure 4.1.: The first two phases of Hooke-Jeeves method.

the loop.

4.3.2. Acceleration operator

The acceleration operator is based on the Hooke-Jeeves method which is a derivative-free method. The Hooke-Jeeves method includes two types of search: exploratory search and pattern search. The first two iterations of the procedure are illustrated in Figure 4.1. Given x_1, an initial exploratory search along the coordinate directions produces a point x_2 (set as y). A pattern search along direction $x_2 - x_1$ leads to a point x'_1. Another exploratory search from x'_1 gives a point x'_2 (set as y'). The next pattern search is conducted along the direction $y' - y$, yielding x''_1. The process is then repeated. An acceleration operator based on the Hooke-Jeeves method is illustrated as follows.

Step 1: Input the starting point x_0, an initial step length t_0 and a tolerance parameter ϵ.

Step 2: Do initial exploratory search: starting from $x_1 (= x_0)$, run line search along the coordinate axes with the initial step length t_0, set the point obtained as x_n and the direction $d = x_n - x_1$. Let $y = x_n$.

Step 3: Do pattern search: starting from y, run a line search along the direction d with an initial step size t_0, set the obtained point as x_1.

Step 4: Exploratory search: starting from x_1, using initial step length t_0, run a line search along the coordinate axes. Set point obtained as x_n and denote the direction $d = x_n - y$. let $y = x_n$.

Step 5: If $|f(y) - f(x_n)| < \epsilon$, then stop; otherwise, go back to Step 3.

Line search plays an extraordinarily important role in the Hooke-Jeeves method. It is required both in pattern search and exploratory search. In general, an optimal line search is applied in the Hooke-Jeeves method, but this may cause some technical issues for problems whose derivative is time consuming or impossible to achieve. Furthermore, optimal line search is time consuming itself and not global convergence. So in our simulations, we use discrete step length to simplify the computational process and avoid the computation of derivative. More precisely, a double step size strategy is introduced instead of optimal line search. This method starts from a small given step size, if the current step size decrease objective function value along the considered direction, we accept it and further test its double; otherwise, we stop line search and take the last accepted step size as a solution.

The initial step size at each generation is chosen according to the following rule:

$$t_0 = \alpha \min_{x_i, x_j \in P} \parallel x_i - x_j \parallel,$$

where $\alpha \in [0, 1]$ is a parameter and P is the current generation. Clearly, the choice of initial step size is self-adaptive in this strategy. At the early stage, the diversity of population is large. So the step sizes should be bigger to guarantee enough decrease of the objective function value. At the latter stage, the population gradually converges to an approximate solution, the decrease of objective function becomes tiny vibration of individuals, which needs the step size to be smaller.

4.3.3. Algorithm: GAHJ

By Embedding the acceleration operator to the general process of genetic algorithm, we have an accelerated genetic algorithm. We call it GAHJ in abbreviation. The acceleration operator can generates some outstanding points in the each generation. However, if the acceleration operator is frequently called during the iterations, the process becomes time consuming and a lot of calculations are actually needless although it can provide more better chromosomes to the next generation. Thus, the acceleration operator should be applied as less as possible. In GAHJ, we run acceleration operator when the current generation decreases the objective function value, i.e., the best point of the current generation is smaller than the current best point.

For the selection operator, we keep the better chromosomes to the next generation so as to guarantee the local exploitation. On the other hand, it is still very important to maintain the diversity of the next generation which guarantees the global exploration. Therefore, Instead of keeping all the better points in the next generation or randomly choosing points to the next generation, we build it by half choosing from the best chromosomes and half choosing randomly. In the following, we present the pesudocode of the GAHJ method in which $P(t)$ and $O(t)$ stand for parents and offspring in the t^{th} generation, respectively.

Algorithm: GAHJ

```
1 Initialization
    1.1 Generate the initial population $P(0)$,
    1.2 Set crossover rate, mutation rate, and maximal generation number,
    1.3 Set $t \leftarrow 0$.
2 While generation counter have not reach the maximal generation number , do
    2.1 Arithmetic crossover operator: generate $O_c(t)$,
    2.2 Nonuniform mutation operator: generate $O_m(t)$,
    2.3 Construct the selection pool by $O(t) = O_c(t) \cup O_m(t)$ and the offspring: compute
        their value of fitness function,
    2.4 Selection operator: build the next generation by choosing half population size of
        best chromosomes and half population size of random chromosomes from $O(t)$,
        respectively.
    2.5 Acceleration operator: update the best point so far, if the best objective function
        value decreases, then acceleration operator is activated with the updated best
        point as s starting point.
    2.6 Set $t \leftarrow t + 1$, go to 2.1
        end
end
```

4.3.4. Parameters

The choice of parameters is important to numerical performance of the genetic algorithm [23], which is the same in GAHJ. For GAHJ, because of the acceleration operator, generation time and population size can be dramatically reduced. Especially for a convex optimization problem, if acceleration operator is activated at a proper starting point, the optimal solution can be obtained in just a few generations. In our simulations, the parameters of GAHJ are various for different experimental tests. Empirically, if the dimension of the problem is n, then, the population size is $2n \sim 5n$, maximal generation number is $20n \sim 50n$. Crossover rate and mutation rate are $0.4 \sim 0.5$ and $0.1 \sim 0.2$, respectively.

4.4. Numerical experiments

In this section, we investigate the numerical performance of GAHJ by testing some constrained optimization benchmarks (see Appendix C). We first implement the GAHJ using the two different penalty models, i.e., quadratic penalty function model (Model 1, the Equation 4.5) and exact penalty function model (Model 2, the Equation 4.6), respectively. Then, numerical comparisons of GAHJ with other constrained global optimization methods are presented.

All test problems are solved in an environment of MATLAB(2010a) installed on an ACER ASPIRE4730Z laptop with a 2G RAM and a 2.16GB CPU.

The test problems are taken from [19]. The main characteristics of these test cases are reported in Table 4.1. These problems include not only different types of objective functions (e.g., linear, nonlinear and quadratic), but also a wide range of constraint functions (e.g., linear inequality (LI), nonlinear equality (NE), and nonlinear inequality (NI)). The feasible ratio ρ is determined by calculating the percentage of feasible solutions among 1,000,000 randomly generated points in the box constraint, i.e.,

$$\rho = \Omega / M,$$

where $M = 1,000,000$ is the number of points randomly generated from X, Ω is the number of feasible points among them. It can be observed that the feasible ratio of Problem 4.3, 4.5, 4.11, 4.13 are all zero. This is because only equality constraints applied in these problems. However, in Problem 4.2 almost all points are feasible (99.9962%).

In order to measure the success rate of all algorithms, we introduce the following criteria,

$$\frac{f^* - f^{**}}{|f^{**}| + 1} < \varepsilon, \tag{4.9}$$

Table 4.1.: Main characteristics of 13 benchmark test functions

Problem	Dim	Type of f	Feasible ratio	LI	NE	NI	f^{**}
Problem 4.1	13	Quadratic	0.0003%	9	0	0	-15.0000
Problem 4.2	20	Nonlinear	99.9962%	1	0	1	-0.803619
Problem 4.3	10	Nonlinear	0.0000%	0	1	0	-1.0000
Problem 4.4	5	Quadratic	26.9557%	0	0	6	-30665.539
Problem 4.5	4	Nonlinear	0.0000%	2	3	0	5126.498
Problem 4.6	2	Nonlinear	0.0053%	0	0	2	-6961.814
Problem 4.7	10	Quadratic	0.0002%	3	0	5	24.306
Problem 4.8	2	Nonlinear	0.8587%	0	0	2	-0.095825
Problem 4.9	7	Nonlinear	0.5182%	0	0	4	680.630
Problem 4.10	8	Linear	0.0005%	3	0	3	7049.248
Problem 4.11	2	Quadratic	0.0000%	0	1	0	0.750
Problem 4.12	3	Quadratic	0.0197%	0	0	9^3	-1.0000
Problem 4.13	5	Nonlinear	0.0000%	0	3	0	0.0539498

and

$$\frac{F^* - f^{**}}{|f^{**}| + 1} < \varepsilon, \tag{4.10}$$

where f^* , F^* and f^{**} are the optimal value of objective function, optimal value of auxiliary function and the current known best optimal solution, respectively. ε is a threshold number which, in our test problems, is $10^{-2} \sim 10^{-3}$. We use the criteria (4.9) and (4.10) together because they guarantee the optimal solution of both the original problem and auxiliary problem. For each test case, 100 independent trials of GAHJ on both Model 1 and Model 2 are simulated. In order to compare the numerical performance of GAHJ on Model 1 and Model 2, we use the same parameters for both models. More specifically, the maximal generation time is set as $400n$; population size is set as $20n$; crossover rate and mutation rate are set as 0.7 and 0.1, respectively.

Comparisons between different penalty models

Table 4.2 shows the success rate of solving the test problems by GAHJ applying Models 1 and 2, respectively. In this table, μ and s are penalty parameter and success rate, respectively.

Table 4.2.: Success rate of GAHJ

Problem	Dim.	Model 1		Model 2	
		μ	s	μ	s
Problem 4.1	13	10^5	100%	10^5	100%
Problem 4.2	20	10	16%	10^4	0%
Problem 4.3	10	10^4	100%	10^4	100%
Problem 4.4	5	10^3	100%	10^5	36%
Problem 4.5	4	100	100%	100	32%
Problem 4.6	2	10^4	100%	10^4	100%
Problem 4.7	10	10^5	100%	100	24%
Problem 4.8	2	10^6	90%	10^6	79%
Problem 4.9	7	10^3	100%	10^6	100%
Problem 4.10	8	10^5	100%	10^6	12%
Problem 4.11	2	10^4	100%	10^6	39%
Problem 4.12	3	10^6	100%	10^5	100%
Problem 4.13	5	10	100%	10^3	100%

This table clearly shows that quadratic penalty transformation of constraints (Model 1) enjoys a better success rate than that of obtained by exact penalty transformation of constraints (Model 2). However, Problem 4.2 is an exception. The exterior penalty function model is suffered a low success rate (16%). Furthermore, the exact penalty function model even failed to obtain an optimal solution. This was also observed in other papers including [120]. The success rate for Problem 4.8 using Model 2 is 90% which is less than others. One can see that the penalty parameter in Model 1 is smaller than that for Model 2.

Table 4.3 shows statistical results for Problems 4.1-4.13 solved by GAHJ using Models 1 and 2, respectively. In this table, we list the best function values (f^*), average function values ($\bar{f}$), and the worst function values ($\tilde{f}$) out of 100 independent runs. The corresponding penalty function values ($p^*,\ \bar{p},\ \tilde{p}$) are also presented. The last column is the standard deviation (S.D.) of function value out of 100 independent executions. Results from this table show that GAHJ achieved good results for all problems except Problem 4.2. Among them, Problems 4.1, 4.6, 4.10 and 4.13 are better solved by exact penalty transformation of constraints (Model 2); Problems 4.3, 4.4, 4.5, 4.7, 4.9 and 4.11 are better solved by quadratic

Table 4.3.: Numerical results of Problem 1-13 solved by GAHJ

Problem	Model	Best		Mean		Worst		S.D.
		f^*	p^*	$\bar{f}$	$\bar{p}$	$\tilde{f}$	$\tilde{p}$	
Problem 4.1	Model 1	**-15.000**	1.4975e-11	-14.968331	1.5500e-11	-13.828132	3.3923e-11	2.8e-14
	Model 2	**-15.000**	0	**-15.000**	0	**-15.000**	0	0
Problem 4.2	Model 1	-0.6114226	4.8896e-15	**-0.5153114**	5.7896e-16	**-0.3924318**	0	3.2e-2
	Model 2	**-0.6238671**	3.8964e-15	-0.4936823	2.7368e-16	-0.3492111	-1.0225e-16	5.4e-2
Problem 4.3	Model 1	**-1.0031596**	1.5958e-9	**-1.0031321**	1.5673e-9	**-1.0030899**	1.5250e-9	9.5e-6
	Model 2	-0.999823	9.6033e-14	-0.9939532	1.0173e-11	-0.9814305	5.6362e-12	5.4e-3
Problem 4.4	Model 1	**-30665.547**	8.8329e-11	-30605.395	1.3680e-11	-30369.105	0	19
	Model 2	-30665.541	3.2513e-8	**-30665.539**	2.3496e-9	**-30665.537**	1.3853e-9	5.3e-4
Problem 4.5	Model 1	**5126.1615**	0.1704e-4	**5127.8993**	0.1674e-4	**5160.0261**	0.1915e-4	4.4
	Model 2	5126.5028	7.5138e-4	5139.9054	3.1882e-3	5173.3279	1.1084e-3	13
Problem 4.6	Model 1	-6961.8121	0	**-6961.6407**	4.7569e-4	-6960.9237	0	1.5
	Model 2	**-6961.8134**	0	-6961.6189	7.1054e-16	**-6961.0593**	0	0.16
Problem 4.7	Model 1	**24.303714**	1.4521e-6	**24.308487**	1.3796e-6	**24.349013**	1.2417e-6	9.4e-3
	Model 2	24.318635	0	24.422653	0	24.556940	0	6.7e-2
Problem 4.8	Model 1	**-0.0958250**	0	**-0.0958250**	0	-0.0958250	0	2.1e-10
	Model 2	**-0.0958250**	0	**-0.0958250**	0	**-0.0958250**	0	1.5e-10
Problem 4.9	Model 1	**680.55806**	3.6152e-3	**680.55836**	3.5841e-3	**680.55860**	3.5606e-3	1.1e-4
	Model 2	680.66750	0	681.37301	0	686.35024	0	0.7
Problem 4.10	Model 1	7053.0718	6.5444e-10	7090.2136	1.1279e-9	7118.7250	0	19
	Model 2	**7052.8082**	0	**7085.3769**	6.8148e-6	**7113.0948**	0	18
Problem 4.11	Model 1	**0.7500150**	1.8339e-9	0.7551281	5.4956e-9	0.7673083	2.3549e-9	4.8e-3
	Model 2	0.7503593	1.7071e-11	**0.7533152**	2.3821e-9	**0.7604427**	3.5e-3	5.3e-4
Problem 4.12	Model 1	-1	0	-1	0	-1	0	0
	Model 2	-1	0	-1	0	-1	0	0
Problem 4.13	Model 1	0.0540487	1.1889e-5	0.0562050	3.6475e-5	0.0639856	1.4242e-4	2.7e-3
	Model 2	**0.0539679**	3.8797e-5	**0.0561241**	3.9393e-5	**0.0629757**	1.7103e-5	2.2e-3

penalty transformation (Model 1); and the same solutions are achieved for Problem 4.8 and 4.12. It is important to note that this comparison is done without taking into account the penalty term. Technically, some of the solutions are corresponding to infeasible points, but very close to feasible region (the degree of closeness can be seen from p^*, $\bar{p}$ and $\tilde{p}$). Since we use exterior penalty approach to handle the constraints, this difficulty cannot be removed in general. With respect to the standard deviation, Model 2 is better than Model 1 for Problems 4.1, 4.4, 4.6, 4.8, 4.10, 4.11 and 4.13, which means that Model 2 is more robust than Model 1 for these problems. However, Model 1 is more stable than Model 2 for Problems 4.2, 4.3, 4.5, 4.7 and 4.9. For Problem 4.12, Model 1 and Model 2 share the same robustness.

Figure 4.2 illustrates the variation of the objective function values of Problems 4.1 and 4.12 when solved using both models. For Problem 4.1, the objective function values are actually smaller than the real optimal function values at the early stage. This is because the search starts from the outside of the feasible region. In this case, the value of the penalty function multiplied by penalty parameter is very large, which enlarges the value of the auxiliary function. Whereas, for Problem 4.12, most of the search appears at the inside of feasible region, which means that the penalty function term is zero. So, value of auxiliary function equals the value of objective function. In both problems, the results obtained by solving Model 1 are better than those obtained by Model 2.

Comparison with FSA method

The FSA [43] is a hybrid method to solve the constrained global optimization problems. In the FSA, the simulated-annealing is used for the global search and a direct search method is used for the local search. Thus, it is highly similar to our method. In [43], some benchmarks are tested by the FSA and the results show that the FSA is much better than some global optimization methods, such as the Homomorphous Mappings method [65], Stochastic Ranking method [101], Adaptive Segregational Constraint Handling EA method [10] and Simple Multimembered Evolution Strategy method [82]. In view of its superiority and the similar feature to our method, we will compare our proposed method with the FSA in this subsection. To apply the FSA, a constrained optimization problem in [43] is first reformulated as a form of optimizing two functions, the objective function and the constraint violation function. Then, the FSA method is applied to solve the reformulated problem. Another common ground for the GAHJ and FSA is that they are both derivative-free. More specifically, their requirements for objective functions and constraint functions are only Lipschitz continuity. The main difference between the GAHJ and FSA is that the GAHJ is a population-based method since it is based upon genetic algorithm, while the FSA is a point-to-point method since it is based upon the simulated annealing. Another difference is the reformulation of the constrained optimization problem. Instead of using penalty function method, the constraints in [43] are reformulated as a nonnegative objective function called constrained violation function. If a point is feasible, the value of the constrained violation function equals to 0.

Table 4.4.: Comparison between GAHJ and FSA

Problem	Optimal solution	FSA	GAHJ
Problem 4.1	Best	-14.999	-15.000
	Worst	-14.980	-15.000
Problem 4.2	Best	-0.7549125	-0.62386713
	Worst	-0.2713110	-0.3492111
Problem 4.3	Best	-1.0000015	-1.0031596
	Worst	-0.9915186	-1.0030899
Problem 4.4	Best	-30665.5380	-30665.541
	Worst	-30664.6880	-30665.537
Problem 4.5	Best	5126.4981	5126.1615
	Worst	5126.4981	5160.0261
Problem 4.6	Best	-6961.81388	-6961.8134
	Worst	-6961.81388	-6961.0593
Problem 4.7	Best	24.310571	24.303714
	Worst	24.644397	24.349013
Problem 4.8	Best	-0.095825	-0.095825041
	Worst	-0.095825	-0.095825040
Problem 4.9	Best	680.63008	680.55806
	Worst	680.69832	680.55860
Problem 4.10	Best	7059.86350	7052.8082
	Worst	9398.64920	7113.0948
Problem 4.11	Best	0.7499990	0.75001507
	Worst	0.7499990	0.76730836
Problem 4.12	Best	-1.0000	-1.0000
	Worst	-1.0000	-1.0000
Problem 4.13	Best	0.0539498	0.053967965
	Worst	0.4388511	0.062975713

Otherwise, it is positive. Under this strategy, the original constrained global optimization problem has been transformed into a multi-objective optimization problem.

The data of the FSA in Table (4.4) and (4.5) is taken from Table 2 of [43] and the data of GAHJ is taken from the better one of Model 1 and Model 2.

Table 4.4 illustrated the best optimal solutions and the worst optimal solutions obtained by FSA and GAHJ. From the data showed in the table, for Problem 4.8 and 4.12, FSA and GAHJ obtained the same results, which are the best known optimal solutions. For Problems 4.1, 4.4, 4.5, 4.6 ,4.7 ,4.9 and 4.10, all of the best and the worst optimal solutions obtained by GAHJ are better than those obtained by FSA. For Problem 4.3, the best optimal solution obtained by GAHJ is -1.0031596 which is not as good as FSA (-1.0000015), but the worst optimal solution obtained by GAHJ is -1.0030899 which is much better than FSA (-0.9915186). For Problem 4.2, FSA performs better at the best optimal solution, but weaker at the worst optimal solution. For Problem 4.13, there is a big gap between the best and worst optimal

Table 4.5.: Statistic comparison between GAHJ and FSA

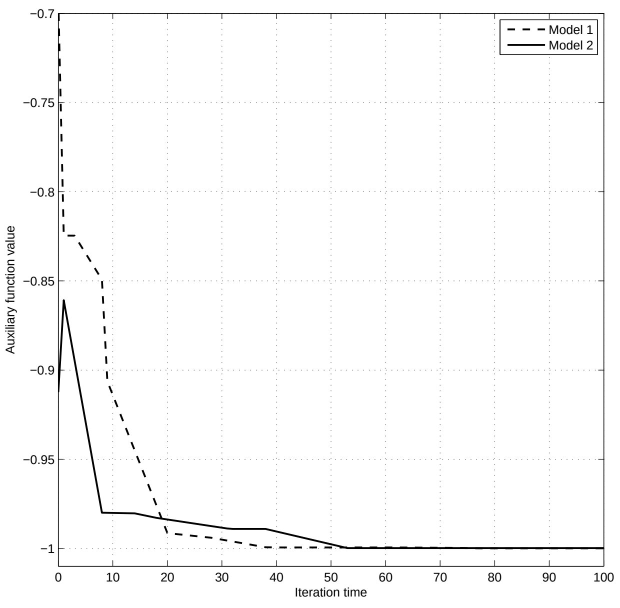
Problem	Statistic features	FSA	GAHJ
Problem 4.1	Mean	-14.993316	-15.000
	S.D.	0.004813	0.000000
Problem 4.2	Mean	0.3717081	-0.5153114
	S.D.	0.098023	0.032000
Problem 4.3	Mean	-0.9991874	-1.0031321
	S.D.	0.001653	0.000009
Problem 4.4	Mean	-30665.4665	-30665.539
	S.D.	0.173218	0.0000530
Problem 4.5	Mean	5126.4981	5127.8993
	S.D.	0.000000	4.4
Problem 4.6	Mean	-6961.81388	-6961.6407
	S.D.	0.000000	1.5
Problem 4.7	Mean	24.3795271	24.308487
	S.D.	0.071635	0.009400
Problem 4.8	Mean	-0.095825	-0.0958250
	S.D.	0.000000	0.000000
Problem 4.9	Mean	680.63642	680.55836
	S.D.	0.014517	0.000110
Problem 4.10	Mean	7509.32104	7085.3769
	S.D.	542.3421	18
Problem 4.11	Mean	0.7499990	0.7533152
	S.D.	0.000000	0.00053
Problem 4.12	Mean	-1.0000	-1.0000
	S.D.	0.000000	0.000000
Problem 4.13	Mean	0.2977204	0.0561241
	S.D.	0.188652	0.002200

solution obtained by FSA. The worst optimal solution obtained by GAHJ is much more better than that obtained by FSA.

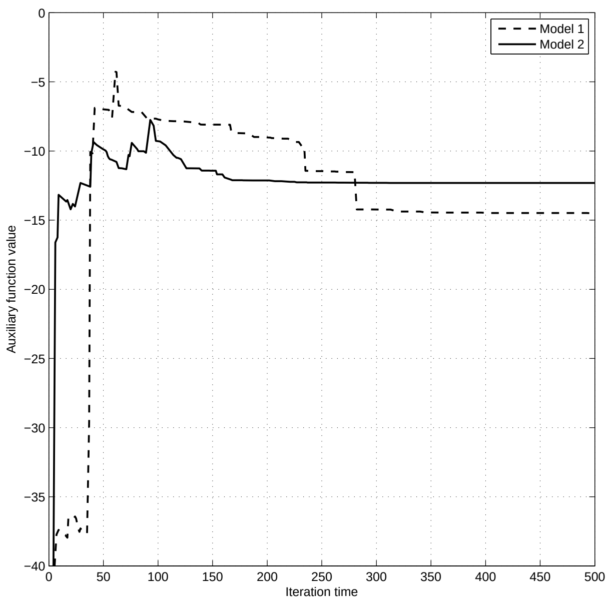
In order to investigate the robustness of GAHJ, we compare its statistical performances with those of FSA. Table 4.5 reports mean value (Mean) and standard deviation (S.D.) of optimal solutions for both GAHJ and FSA. Table 4.5 shows that GAHJ achieves better mean values for Problems 4.1, 4.2, 4.3, 4.4, 4.7, 4.9, 4.10, 4.11 and 4.13. For Problem 4.10, the mean value of FSA is 7509.32104, while that of GAHJ is 7085.3769 which is much smaller than that of FSA. For Problem 4.13, the mean value of FSA is 0.2415963, while that of GAHJ is 0.0539498. FSA achieves a better mean value than GAHJ only for Problems 4.5, 4.6 and 11. On the other hand, from data of standard deviation, the optimal solutions obtained by GAHJ are more robust than those obtained by FSA. Except Problems 4.5, 4.6 and 4.11, solutions obtained by GAHJ have smaller standard deviation than those obtained by FSA, which means that GAHJ is more robust than FSA. GAHJ and FSA share the same statistical performances for Problems 4.8 and 4.12. The above analysis clearly shows that as a global optimization solver, GAHJ is much more superior than FSA.

4.5. Conclusions

In this chapter, we developed a new hybrid method for solving constrained global optimization problems. This method is based upon the combination of the genetic algorithm and Hooke Jeeves method. The Hooke Jeeves method was embedded into the genetic algorithm as an acceleration operator during the iterations. The numerical experiments show that the proposed hybrid method achieves better performances than the FSA. However, since we use exterior penalty function method to handle the constraints, some of the optimal solutions obtained by the proposed method may be infeasible. Therefore, our future work for this subject is to improve the constraint handling technique to ensure the feasibility of the obtained solution. A possible strategy is to introduce the greedy selection (death penalty). This strategy excludes all the infeasible candidate solutions gradually.



(a) Problem 1



(b) Problem 12

Figure 4.2.: The Objective function value versus iteration time.

Chapter 5.

A method for nonsmooth equation system

In this chapter, we consider the following system of equations

$$H(x) = \begin{pmatrix} H_1(x) \\ H_2(x) \\ \vdots \\ H_m(x) \end{pmatrix} = 0, \tag{5.1}$$

where $H_1, H_2, \cdots, H_m : \mathbb{R}^n \to \mathbb{R}$ are locally Lipschitz continuous but not necessarily differentiable. We call Problem 5.1 the *system of nonsmooth equations*.

5.1. Introduction

The system of nonsmooth equations appears in many applications, such as nonlinear complementarity, variational inequality and bilevel programming problems. Among the different methods for solving the system of nonsmooth equations, Newton's method is one of the most widely used ones. (see [49, 50, 88, 97, 98]). Newton's method originally developed for solving the continuously differentiable system of equations. In nodifferentiable case the main obstacle is the choice of the generalized Jacobian. Because for the nonsmooth functions the generalized Jacobian is not unique and it is not easy to compute it. To overcome this difficulty, smooth approximations of the generalized Jacobian have been used by many authors. Then a system of nonsmooth equations was solved by semismooth Newton's method or smoothing Newton's method [96]. Some typical semismooth Newton's methods refer to [54, 87, 106] and some smoothing Newton methods can be found in [22, 21, 33, 52, 53, 95]. Another numerical approach for solving the system of nonsmooth equations is the quasi-Newton method. In [112], the authors presented a unified realization of quasi-Newton methods for solving several standard problems including the complementarity problem, variational inequality and KKT system. The algorithm proposed in [112] solves only a system of linear equations at each iteration. In [74], the BFGS method is adopted to solve the KKT system. The KKT system is first reformulated into a system of nonsmooth equations, then the system of nonsmooth equations is split successively into subproblems such that each of them has a particular structure, and finally the BFGS algorithm is applied to solve those subproblems. Some other discussions about solving the system of nonsmooth equations refer to [31, 91, 94].

Actually, a system of equations, whether smooth or nonsmooth, is equivalent to an unconstrained optimization problem and if the system is consistent then the optimization problem has a zero global minimum value. So, solving the system of nonsmooth is equivalent to (5.1) equals to solve the unconstrained optimization problem

$$\begin{cases} \text{Minimize} & \| H(x) \| \\ \text{Subject to} & x \in \mathbb{R}^n \end{cases} \tag{5.2}$$

or

$$\begin{cases} \text{Minimize} & \frac{1}{2} \parallel H(x) \parallel^2 \\ \text{Subject to} & x \in \mathbb{R}^n \end{cases}. \tag{5.3}$$

Here $\parallel \cdot \parallel$ stands for the Euclidian norm. This strategy is applied in [125] where a filled function method is used to solve Problem (5.3). If $H_i(x)$ is nonsmooth, the nonsmoothness may still appear in solving Problem (5.2) and (5.3). In this case, the gradient-based methods cannot be applied. To overcome this difficulty, we apply the nonsmooth optimization method, such as the quasisecant method, to solve Problem (5.2) or (5.3).

5.2. An algorithm for the system of nonsmooth equations

As discussed before, solving the system of nonsmooth equations $H(x) = 0$ is equivalent to solve an optimization problem whose optimal solution is zero if this system is consistent, i.e., Problem (5.2) or (5.3). In this section, we use the quasisecant method to solve the optimization problem 5.3. Denote

$$f(x) = \frac{1}{2} \parallel H(x) \parallel^2 = \frac{1}{2} \sum_{i=1}^{m} H_i^2(x).$$

Then, the optimization problem (5.3) can be rewritten as

$$\begin{cases} \text{Minimize} & f(x) \\ \text{Subject to} & x \in \mathbb{R}^n \end{cases}. \tag{5.4}$$

First, we discuss how to compute the quasisecant $v(x, g, h)$ of the function f at the point x along the direction of g with length $h > 0$.

5.2.1. Algorithms

Theorem 5.2.1 Suppose that f is convex function, if $v \in \partial_c f(x + hg)$, then $v = v(x, g, h)$, i.e., v is a quasisecant of f along direction g with length $h > 0$.

Proof: From $v \in \partial_c f(x + hg)$, we have, for any $x' \in \mathbb{R}^n$,

$$f(x') \geq f(x + hg) + \langle v, x' - x - hg \rangle.$$

Let $x' = x$, then

$$f(x) \geq f(x + hg) + \langle v, -hg \rangle$$

which means that $v = v(x, g, h)$. ■

Theorem 5.2.1 gives a sufficient condition to compute quasisecant of function f. A quasisecant of f at x along direction g with length $h > 0$ can be computed as a subgradient of f at the point $y = x + hg$. More specifically, if $f(x)$ is continuously differentiable, we have

$$\begin{aligned} v(x, g, h) &= \nabla f(x + hg) = \nabla f(y) \\ &= H_1(y)\nabla H_1(y) + H_2(y)\nabla H_2(y) +, \ldots, H_m(y)\nabla H_m(y) \\ &\triangleq \left[(H_1(y), H_2(y), \ldots, H_m(y)) \begin{pmatrix} \nabla H_1^T(y) \\ \nabla H_2^T(y) \\ \vdots \\ \nabla H_m^T(y) \end{pmatrix} \right]^T . \end{aligned}$$

The continuous differentiability requirement of $f(x)$ is strict, since it needs continuous differentiability of each $H_i(x)$. If some of the $H_i(x)$ are nonsmooth convex functions, maximum functions or D.C. functions, their subgradient can be calculated by using algorithms proposed in [3, 5]. Furthermore, if $f(x)$ can be represented as a D.C. function, i.e., $f(x) = g_1(x) - g_2(x)$, where $g_1(x)$ and $g_2(x)$ are both convex functions, then the quasisecant of $f(x)$ can be implicitly computed by subgradients of $g_1(x)$ and $g_2(x)$ [2, 4].

The algorithm of the quasisecant method for the system of nonsmooth equations is composed by two steps: inner iteration and outer iteration. For given $h > 0$, the inner iteration is to find a descent direction in the form of $x + hg$, where $g \in S_1$. The length $h > 0$ in inner iteration controls accuracy of approximation of $W(x, h)$. In general, the smaller h is, the better approximation of subdifferential it produces. The outer iteration is to adjust the length h. When no descent direction is found or an approximate stationary point is obtained in inner iteration, h is decreased by executing the outer iteration. This means when inner iteration can not get any decrease, we decrease the length h in order to get a better approximation of $W(x, h)$.

Algorithm 5.2.1 Inner iteration

Step 0: Input data: the current point x_k from outer iteration and set it as a starting point, i.e., $x := x_k$; the current length $h_k > 0$ and tolerance $\delta_k > 0$ from outer iteration, respectively; set $c_1 \in (0, 1)$ and the maximum number of iterations $m > 0$.

Step 1: Initialization: choose any unit direction $g^1 \in S_1$ and compute a quasisecant $v^1 \in \partial f(x + h_k g^1)$. Set $V_1(x) := \{v^1\}$ and an inner iteration counter $j := 1$.

Step 2: Solve the subproblem

$$\begin{cases} \min & \| \xi \|^2 \\ \text{s.t.} & \xi \in \text{co}V_j(x). \end{cases} \tag{5.5}$$

Let ξ^* to be the solution of (5.5). ξ^* is the nearest point of co$V_j(x)$ to the origin of $\mathbb{R}^n$..

Step 3: If $\| \xi^* \| \leq \delta_k$, then stop the inner loop; otherwise, reverse and normalize vector ξ^*, i.e.,

$$g^{j+1} = -\frac{\xi^*}{\| \xi^* \|}.$$

Step 4: If

$$f(x + h_k g^{j+1}) - f(x) \leq -c_1 h_k \parallel \xi^* \parallel, \tag{5.6}$$

then go to Step 6. Otherwise go to Step 5.

Step 5: If $j > m$, then stop the inner loop; otherwise compute a quasisecant in direction g^{j+1}, $v^{j+1} \in \partial f(x + h_k g^{j+1})$ and construct set $V_{j+1}(x) := V_j(x) \cup \{v^{j+1}\}$. Set $j := j + 1$ and go to Step2.

Step 6: Line search: run double step size line search along the direction g^{j+1} starting from x. Let the result of line search to be x^*, then assign x^* to x, i.e., $x := x^*$ and go to Step 1.

Some explanations of Algorithm 5.2.1 are as follows. According to Theorem 5.2.1, the quasisecant of $f(x)$ can be explicitly calculated using the subdifferential. Thus all the quasisecants are computed as subgradients of $f(x)$ at $x + h_k g^j$. Note that h_k is fixed in the inner iteration, so these points are actually distributed on a circle centered at x with radius h_k.. In Step4, as long as g^{j+1} is verified as a descent direction, i.e., condition (5.6) is satisfied, an inexact line search runs along this direction in Step 6. We use double step size strategy (see Page 66) which was presented in the quasisecant method in inexact line search. There are two stopping criteria in the inner loop: in Step 3, if $\parallel \xi^* \parallel \leq \delta_k$, an $(h_k, \delta_k)-$stationary point is obtained; in Step 5, the maximum number of iterations is reached, which means that a smaller h is needed to improve the approximation of the subdifferential.

Algorithm 5.2.2 Outer iteration

Step 0: Input data: a starting point x_0, parameters $h_0 > 0$, $\delta_0 > 0$, and set a counter $k := 0$.

Step 1: If $f(x_k) = 0$, then stop the outer loop and x_k is an optimal solution of Problem (5.4), i.e., a solution of the system (5.1). Otherwise, if $0 \in \partial f(x_k)$, then stop the outer loop, the system is inconsistent and let x_k as an approximate solution of the system.

Step 2: Let $h_{k+1} = \frac{1}{2}h_k$ and $\delta_{k+1} = \frac{1}{2}\delta_k$.

Step 3: Run the inner iteration (Algorithm 5.2.1) by inputting the current point x_k, length h_k and tolerance δ_k. Let x_{k+1} be the output of inner iteration.

Step 4: Set $k := k + 1$ and go to Step 1.

There are two stopping criteria for the outer iteration, i.e. $0 \in \partial f(x_k)$ and $f(x_k) = 0$. If $0 \in \partial f(x_k)$, then x_k is a Clarke stationary point which may not be a solution of (5.4). If $f(x_k) = 0$, then x_k is a solution of (5.4).

5.2.2. Convergence analysis

Lemma 5.2.1 Suppose that $f_i(x),\ i = 1, \cdots, m$ are all Lipschitz functions and bounded by M, i.e., $|f_i(x)| \leq M,\ i = 1, \cdots, m$. Let

$$f(x) = \sum_{i=1}^{m} f_i^2(x),$$

then $f(x)$ is a Lipschitz function.

Proof: Since $f_i(x),\ i = 1, \cdots, n$ are all Lipschitz functions, there exists $\alpha_i,\ i = 1, \cdots, m$, such that

$$|f_i(x_1) - f_i(x_2)| \leq \alpha_i \parallel x_1 - x_2 \parallel,\ i = 1, \cdots, m,$$

for all x_1 and x_2. Let $\alpha = \max\{\alpha_i | i = 1, \cdots, n\}$. For any x_1 and x_2,

$$\begin{aligned} |f(x_1) - f(x_2)| &= |\sum_{i=1}^{m} f_i^2(x_1) - \sum_{i=1}^{m} f_i^2(x_2)| \\ &= |\sum_{i=1}^{m}(f_i^2(x_1) - f_i^2(x_2))| \\ &\leq \sum_{i=1}^{m} |(f_i(x_1) + f_i(x_2))(f_i(x_1) - f_i(x_2))| \\ &\leq 2M \sum_{i=1}^{m} \alpha_i \parallel x_1 - x_2 \parallel \\ &\leq 2mM\alpha \parallel x_1 - x_2 \parallel . \end{aligned}$$

This proves that $f(x)$ is Lipschitz continuous function.■

For the point $x_0 \in \mathbb{R}^n$, we consider the set $\mathcal{L}(x_0) = \{x \in \mathbb{R}^n | \, f(x) \leq f(x_0)\}$.

Theorem 5.2.2 Suppose that the system of nonsmooth equations (5.1) is consistent and functions $H_i(x), \; i = 1, \cdots, n$ are all bounded Lipschitz continuous. Suppose the set $W(x, h)$ is constructed using SR-quasisecants and the set $\mathcal{L}(x_0)$ is bounded for starting point $x_0 \in \mathbb{R}^n$. Assume that $\{x_k\}_{k=1}^{\infty}$ is a sequence obtained by applying Algorithm 5.2.2. Then the accumulation point x^* of sequence $\{x_k\}$ is either a solution of system (5.1) or a Clark stationary point of $\| \, H(x) \, \|^2$.

Proof: It is obvious that $x_k \in \mathcal{L}(x_0)$ for all k. The boundedness of set $\mathcal{L}(x_0)$ implies that the sequence $\{x_k\}$ has at least one accumulation point. Without loss of generality, let x^* be an accumulation point and $x_k \to x^*$ when $k \to +\infty$.

Now, suppose that Algorithm 5.2.2 is not terminated by $0 \in \partial f(x_k)$, because Algorithm 5.2.1 is a decreasing method, then we have

$$f(x_1) \geq f(x_2) \geq \cdots \geq f(x_k) \geq \cdots \geq 0.$$

This is to say, $\{f(x_k)\}_{k=1}^{\infty}$ is a decreasing sequence with a lower boundary 0. Therefore, there exists a subsequence $\{f(x_{k_i})\}$ such that $f(x_{k_i}) \to 0$ when $i \to +\infty$. Note that $x_{k_i} \to x^*$ when $i \to +\infty$, according to the continuity of f, we have

$$f(x^*) = 0,$$

which means Algorithm 5.2.2 converges at a solution of equation system (5.1).

On the contrary, if Algorithm 5.2.2 does not terminate at $f(x_k) = 0$, then note that

$$f(x) = \frac{1}{2} \| \, H(x) \, \|^2 = \frac{1}{2} \sum_{i=1}^{m} H_i^2(x),$$

according to Lemma 5.2.1, $f(x)$ is a Lipschitz function and bounded below, i.e., $f(x) \geq 0$. Therefore, based on Theorem 4.5 in [2], every accumulation point of the sequence $\{x^k\}$ belongs to the set $X^0 = \{x \in \mathbb{R}^n | \ 0 \in \partial f(x)\}$, i.e., x^* is a Clark stationary point of $\| H(x) \|^2$. This completes the proof.■

5.3. Numerical examples

The system of nonsmooth equaitons usually appears in solving mathematical programming problems, such as the bilevel programming and complementarity problems. During the process of solving the bilevel programming problem, we need to transform the lower level problem into the system of nonsmooth equations. Clearly, numerical results of solving bilevel programming problems are heavily dependent on the solution of system of nonsmooth equations. The complementarity problem can also be transformed into the system of nonsmooth equations.

In order to investigate the accuracy of solutions, the following criterion is adopted. Suppose that x^* is a solution, the value of each function in the system of nonsmooth equation is expressed as

$$H(x^*) = (H_1(x^*), H_2(x^*), \cdots, H_m(x^*))^T.$$

Then, the *average error* of all the functions is defined as

$$e = \frac{\sum\limits_{i=1}^{m} |H_i(x^*)|}{m},$$

where $|\cdot|$ stands for absolute value. $H(x^*)$ reflects the accuracy of a solution for each function in the system of nonsmooth equation, while the average error e reveals the whole accuracy of a solution.

The proposed algorithm is tested using some test problems with smooth and nonsmooth system of equations. All test problems are run in an environment of MATLAB(2010a) installed on an ACER ASPIRE4730Z laptop with a 2G RAM and a 2.16GB CPU. To present numerical results we use the following notation:

- k — index for different starting points;
- $\bar{x}$ — parameter from the upper level problem;
- y_0 and λ_0 — starting point;
- y^* and λ^*— solution for the test problems;
- $H(\bar{x}, y^*, \lambda^*)$ — value of equation systems at the solutions;
- e — the average error of function values for solution x^*.

The first three problems are bilevel programming problems which are from [85]. We first illustrate the process of transforming the lower level problems into systems of nonsmooth equations and then use the proposed method to solve the transformed systems of nonsmooth equations. The results are presented in Table 5.2 and compared with the known solution which is presented in Table 5.1.

Example 5.3.1 Consider the bilevel programming problem with the upper-level objective function

$$f(x,y) = \frac{1}{2}(y_1 - 3)^2 + \frac{1}{2}(y_2 - 4)^2$$

and the lower-level optimization problem (in variable y)

$$\begin{cases} \text{Minimize} & \frac{1}{2}[(1+0.2x)y_1^2 + (1+0.1x)y_2^2] - (3+1\frac{1}{3}x)y_1 - xy_2 \\ \text{Subject to} & \\ & \begin{aligned} -\tfrac{1}{3}y_1 + y_2 - 1 + 0.1x &\leq 0 \\ y_1^2 + y_2^2 - 9 - 0.1x &\leq 0 \\ -y_1 &\leq 0 \\ -y_2 &\leq 0, \end{aligned} \end{cases}$$

which is dependent on the parameter $x \in \mathbb{R}$.

Assume that the admissible set $U_{ad} = [1, 10]$. For $x \in U_{ad}$, the lower-level problem can be transformed into the form of *Generalized equation*

$$0 \in \begin{bmatrix} (1+0.2x)y_1 - (3+1\frac{1}{3}x) - \frac{1}{3}\lambda_1 + 2y_1\lambda_2 - \lambda_3 \\ (1+0.1x)y_2 - x + \lambda_1 + 2y_2\lambda_2 - \lambda_4 \\ \frac{1}{3}y_1 - y_2 + 1 - 0.1x \\ -y_1^2 - y_2^2 + 9 + 0.1x \\ y_1 \\ y_2 \end{bmatrix} + N_{\mathbb{R}^2 + \mathbb{R}^4_+(y,\lambda)}, \tag{5.7}$$

which is equivalent to

$$\begin{bmatrix} (1+0.2x)y_1 - (3+1\frac{1}{3}x) - \frac{1}{3}\lambda_1 + 2y_1\lambda_2 - \lambda_3 \\ (1+0.1x)y_2 - x + \lambda_1 + 2y_2\lambda_2 - \lambda_4 \\ \frac{1}{3}y_1 - y_2 + 1 - 0.1x \\ -y_1^2 - y_2^2 + 9 + 0.1x \\ \min\{y_1, \lambda_3\} \\ \min\{y_2, \lambda_4\} \end{bmatrix} = 0. \tag{5.8}$$

Table 5.1.: Known solution for equation system (5.8)

k	$\bar{x}$	y_1	y_2	λ_1	λ_2	λ_3	λ_4
1	4.0604	2.6822	1.4871	0	0.6621	0	0

Table 5.2.: Numerical results for equation system (5.8)

k	$(\bar{x}, y, \lambda)$	$(\bar{x}, y^*, \lambda^*)$	$H(\bar{x}, y^*, \lambda^*)$	e
1	$\begin{pmatrix} 4.0604 \\ 0.3111 \\ 0.9234 \\ 0.4302 \\ 0.1848 \\ 0.9049 \\ 0.9797 \end{pmatrix}$	$\begin{pmatrix} 4.0604 \\ 2.6818 \\ 1.4879 \\ -0.0030 \\ 0.6625 \\ 0.0000 \\ 0.0000 \end{pmatrix}$	$\begin{pmatrix} 0.1121 \\ -0.4816 \\ -0.3940 \\ -0.1561 \\ 0.4723 \\ 0.4112 \end{pmatrix} \times 10^{-7}$	3.3788×10^{-8}
2	$\begin{pmatrix} 4.0604 \\ 0.4389 \\ 0.1111 \\ 0.2581 \\ 0.4087 \\ 0.5949 \\ 0.2622 \end{pmatrix}$	$\begin{pmatrix} 4.0604 \\ 2.6818 \\ 1.4879 \\ -0.0030 \\ 0.6625 \\ -0.0000 \\ 0.0000 \end{pmatrix}$	$\begin{pmatrix} -0.0410 \\ 0.0117 \\ 0.2347 \\ -0.0397 \\ -0.0193 \\ 0.0645 \end{pmatrix} \times 10^{-6}$	6.8469×10^{-8}
3	$\begin{pmatrix} 4.0604 \\ 0.6028 \\ 0.7112 \\ 0.2217 \\ 0.1174 \\ 0.2967 \\ 0.3188 \end{pmatrix}$	$\begin{pmatrix} 4.0604 \\ 2.6818 \\ 1.4879 \\ -0.0030 \\ 0.6625 \\ 0.0000 \\ 0.0000 \end{pmatrix}$	$\begin{pmatrix} -0.3134 \\ -0.1598 \\ 0.9717 \\ 0.0403 \\ 0.2099 \\ 0.1666 \end{pmatrix} \times 10^{-7}$	3.1028×10^{-8}

It is clear that the equation system (5.8) is a system of nonsmooth equations since there are two minimum functions. When parameter from the upper level problem $\bar{x} = 4.0604$, the solution of the system (5.8) is presented in Table 5.1. For this solution, $H(x^*) = (-0.0017, -0.0003,\ 0.0009, 0.0004, 0, 0)$ and the average error $e = 5.4999 \times 10^{-4}$. Table 5.2 presents the numerical solutions obtained by our proposed method. From Table 5.2, the proposed method can achieve the same solution from different starting points generated randomly. By direct comparison, we can see that our solutions are much better than the solutions given in [85].

Example 5.3.2 Consider the bilevel programming problem with the upper-level objective function

$$f(x, y) = x_1^2 - 2x_1 - x_2^2 + y_1^2 + y_2^2$$

and the lower-level optimization problem (in terms of variable y)

$$\begin{cases} \text{Minimize} \quad (y_1 - x_1)^2 + (y_2 - x_2)^2 \\ \text{Subject to} \\ \qquad (y_1 - 1)^2 \leq 0.25 \\ \qquad (y_2 - 1)^2 \leq 0.25 \end{cases}$$

which is dependent on the parameter $x \in \mathbb{R}^2$.

Assume that the admissible set $U_{ad} = [0, 2] \times [0, 2]$. For $x \in U_{ad}$, the lower-level problem can be transformed to the form of *Generalized equation*

$$0 \in \begin{bmatrix} 2(y_1 - x_1) + 2\lambda_1(y_1 - 1) \\ 2(y_2 - x_2) + 2\lambda_2(y_2 - 1) \\ -(y_1 - 1)^2 + 0.25 \\ -(y_2 - 1)^2 + 0.25 \end{bmatrix} + N_{\mathbb{R}^2 + \mathbb{R}^2_+(y,\lambda)}, \tag{5.9}$$

which is equivalent to

$$\begin{bmatrix} 2(y_1 - x_1) + 2\lambda_1(y_1 - 1) \\ 2(y_2 - x_2) + 2\lambda_2(y_2 - 1) \\ \min\{-(y_1 - 1)^2 + 0.25, \lambda_1\} \\ \min\{-(y_2 - 1)^2 + 0.25, \lambda_2\} \end{bmatrix} = 0. \tag{5.10}$$

We solve this system of nonsmooth equations with different starting points $\bar{x}$ (which are generated randomly). The numerical results are presented in Table 5.3. From this table, we can see that for different $\bar{x}$ and randomly generated starting points, our proposed method can solve this system of nonsmooth equations efficiently.

Example 5.3.3 Consider the bilevel programming problem with the upper-level objective

Table 5.3.: Numerical results for equation system (5.9)

k	$(\bar{x}, y, \lambda)$	$(\bar{x}, y^*, \lambda^*)$	$H(\bar{x}, y^*, \lambda^*)$	e
1	$\begin{pmatrix} 0.4324 \\ 0.8253 \\ 0.0835 \\ 0.1332 \\ 0.1734 \\ 0.3909 \end{pmatrix}$	$\begin{pmatrix} 0.4324 \\ 0.8253 \\ 0.5000 \\ 0.8253 \\ 0.1352 \\ -0.0000 \end{pmatrix}$	$\begin{pmatrix} -0.0879 \\ -0.0131 \\ 0.1484 \\ -0.0090 \end{pmatrix} \times 10^{-6}$	6.4601×10^{-8}
2	$\begin{pmatrix} 0.8314 \\ 0.8034 \\ 0.0605 \\ 0.3993 \\ 0.5269 \\ 0.4168 \end{pmatrix}$	$\begin{pmatrix} 0.8314 \\ 0.8034 \\ 0.8314 \\ 0.8034 \\ -0.0000 \\ -0.0000 \end{pmatrix}$	$\begin{pmatrix} -0.1821 \\ -0.1895 \\ -0.0350 \\ -0.0596 \end{pmatrix} \times 10^{-7}$	1.1656×10^{-8}
3	$\begin{pmatrix} 0.6569 \\ 0.6280 \\ 0.2920 \\ 0.4317 \\ 0.0155 \\ 0.9841 \end{pmatrix}$	$\begin{pmatrix} 0.6569 \\ 0.6280 \\ 0.6569 \\ 0.6280 \\ 0.0000 \\ 0.0000 \end{pmatrix}$	$\begin{pmatrix} -0.3011 \\ -0.0193 \\ 0.3880 \\ 0.3182 \end{pmatrix} \times 10^{-7}$	2.5666×10^{-8}

function

$$f(x, y) = 2x_1 + 2x_2 - 3y_1 - 3y_2 - 60 + r[\max\{0, x_1 + x_2 + y_1 - 2y_2 - 40\}]^2,$$

where $r = 100$ is the penalty parameter. The lower-level optimization problem (in terms of variable y)

$$\begin{cases} \text{Minimize} \quad (y_1 - x_1)^2 + (y_2 - x_2)^2 + 40(y_1 + y_2) \\ \text{Subject to} \\ \qquad -y_1 - 10 \le 0 \\ \qquad -y_2 - 10 \le 0 \\ \qquad 2y_1 - x_1 + 10 \le 0 \\ \qquad 2y_2 - x_2 + 10 \le 0 \end{cases}$$

which is dependent on the parameter $x \in \mathbb{R}^2$.

Assume that the admissible set $U_{ad} = [0, 50] \times [0, 50]$. For $x \in U_{ad}$, the lower-level problem can be transformed into the form of *Generalized equation*

$$0 \in \begin{bmatrix} 2(y_1 - x_1) - \lambda_1 + 2\lambda_3 + 40 \\ 2(y_2 - x_2) - \lambda_2 + 2\lambda_4 + 40 \\ y_1 + 10 \\ y2 + 10 \\ -2y_1 + x_1 - 10 \\ -2y_2 + x_2 - 10 \end{bmatrix} + N_{\mathbb{R}^2 + \mathbb{R}^4_+(y,\lambda)}, \tag{5.11}$$

which can be rewritten as

$$\begin{bmatrix} 2(y_1 - x_1) - \lambda_1 + 2\lambda_3 + 40 \\ 2(y_2 - x_2) - \lambda_2 + 2\lambda_4 + 40 \\ \min\{y_1 + 10, \lambda_1\} \\ \min\{y_2 + 10, \lambda_2\} \\ \min\{-2y_1 + x_1 - 10, \lambda_3\} \\ \min\{-2y_2 + x_2 - 10, \lambda_4\} \end{bmatrix} = 0. \tag{5.12}$$

Table 5.4 presents the corresponding numerical results by our method. Among them, the first two results are obtained with $\bar{x} = (0, 0)$. The others are obtained with $\bar{x}$ generated randomly. For all of them, the starting points are generated randomly.

The last problem is a nonlinear complementarity problem from [128]. The strategy for solving the nonlinear complementarity problem is to transform it first into the system of nonsmooth equations (5.1) and then to the nonsmooth optimization problem (5.4).

Example 5.3.4 Consider the following nonlinear complementarity problem: Find $x \in \mathbb{R}^4$ such that

$$x \geq 0, \quad f(x) \geq 0, \quad x^T f(x) = 0$$

Table 5.4.: Numerical results for equation system (5.11)

k	$(\bar{x}, y, \lambda)$	$(\bar{x}, y^*, \lambda^*)$	$H(\bar{x}, y^*, \lambda^*)$	c
1	$\begin{pmatrix} 0 \\ 0 \\ 0.8178 \\ 0.2607 \\ 0.5944 \\ 0.0225 \\ 0.4253 \\ 0.3127 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ -10.0000 \\ -10.0000 \\ 20.0000 \\ 20.0000 \\ 0.0000 \\ -0.0000 \end{pmatrix}$	$\begin{pmatrix} -0.4732 \\ -0.1762 \\ 0.2913 \\ -0.0082 \\ 0.5937 \\ -0.3465 \end{pmatrix} \times 10^{-7}$	3.1488×10^{-8}
2	$\begin{pmatrix} 0 \\ 0 \\ 0.1615 \\ 0.1788 \\ 0.4229 \\ 0.0942 \\ 0.5985 \\ 0.4709 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ -10.0000 \\ -10.0000 \\ 20.0000 \\ 20.0000 \\ -0.0000 \\ -0.0000 \end{pmatrix}$	$\begin{pmatrix} -0.1392 \\ 0.1849 \\ -0.3808 \\ -0.3426 \\ -0.4025 \\ -0.3579 \end{pmatrix} \times 10^{-7}$	3.0134×10^{-8}
3	$\begin{pmatrix} 34.7975 \\ 34.9944 \\ 0.6385 \\ 0.0336 \\ 0.0688 \\ 0.3196 \\ 0.5309 \\ 0.6544 \end{pmatrix}$	$\begin{pmatrix} 34.7975 \\ 34.9944 \\ 12.3987 \\ 12.4972 \\ 0.0000 \\ 0.0000 \\ 2.3987 \\ 2.4972 \end{pmatrix}$	$\begin{pmatrix} -0.3010 \\ -0.2619 \\ 0.0413 \\ 0.4902 \\ -0.1188 \\ -0.1032 \end{pmatrix} \times 10^{-7}$	2.1941×10^{-8}
4	$\begin{pmatrix} 20.3810 \\ 40.9991 \\ 0.7184 \\ 0.9686 \\ 0.5313 \\ 0.3251 \\ 0.1056 \\ 0.6110 \end{pmatrix}$	$\begin{pmatrix} 20.3810 \\ 40.9991 \\ 0.3810 \\ 15.4995 \\ 0.0000 \\ -0.0000 \\ -0.0000 \\ 5.4995 \end{pmatrix}$	$\begin{pmatrix} 0.1841 \\ -0.0711 \\ 0.0203 \\ -0.2368 \\ -0.7016 \\ 0.0609 \end{pmatrix} \times 10^{-7}$	2.1248×10^{-8}

where $f : \mathbb{R}^4 \to \mathbb{R}^4$ is given by

$$\begin{aligned} f_1(x) &= 3x_1^2 + 2x_1x_2 + 2x_2^2 + x_3 + 3x_4 - 6, \\ f_2(x) &= 2x_1^2 + x_1 + x_2^2 + 10x_3 + 2x_4 - 2, \\ f_3(x) &= 3x_1^2 + x_1x_2 + 2x_2^2 + 2x_3 + 9x_4 - 9, \\ f_4(x) &= x_1^2 + 3x_2^2 + 2x_3 + 3x_4 - 3. \end{aligned}$$

This problem is equivalent to solving the nonsmooth equation system

$$F(x) = 0, \quad \text{where } F(x) = \min[f(x), x], \tag{5.13}$$

and $\min$ denotes the componentwise minimum. This problem has two solutions

$$x_1^* = (1, 0, 3, 0)^T, \quad x_2^* = (\sqrt{6}/2, 0, 0, 0.5)^T.$$

Clearly, $F(x)$ is differentiable at x_1^* but nondifferentiable at x_2^*.

In [128], the Newton's method was used to solve the system of nonsmooth equations (5.13). We solve the same problem by the proposed method and the numerical results are given in Table 5.5. Compared with the numerical results presented in Table 2 and Table 3 in [128], our method obtained more exact results than those by the Newton's method.

5.4. Conclusion

This chapter presented a new method for solving the system of nonsmooth equations based upon the quasisecant method. Given the equivalence between the system of nonsmooth equations and the nonsmooth optimization problem, the original system of nonsmooth equations is handled by solving a nonsmooth optimization problem using the quasisecant method. Several numerical experiments are presented to test the proposed method. The numerical results

Table 5.5.: Numerical results for equation system (5.13)

k	$(\bar{x}, y, \lambda)$	$(\bar{x}, y^*, \lambda^*)$	$H(\bar{x}, y^*, \lambda^*)$	e
1	$\begin{pmatrix} 0.6160 \\ 0.4733 \\ 0.3517 \\ 0.8308 \end{pmatrix}$	$\begin{pmatrix} 1.2247 \\ -0.0000 \\ -0.0000 \\ 0.5000 \end{pmatrix}$	$\begin{pmatrix} 0.0188 \\ -0.1921 \\ -0.0932 \\ -0.0428 \end{pmatrix} \times 10^{-6}$	0.0867×10^{-6}
2	$\begin{pmatrix} 0.5853 \\ 0.5497 \\ 0.9172 \\ 0.2858 \end{pmatrix}$	$\begin{pmatrix} 1.2247 \\ -0.0000 \\ -0.0000 \\ 0.5000 \end{pmatrix}$	$\begin{pmatrix} 0.0162 \\ -0.2152 \\ -0.0565 \\ -0.0057 \end{pmatrix} \times 10^{-6}$	0.0734×10^{-6}
3	$\begin{pmatrix} 0.7572 \\ 0.7537 \\ 0.3804 \\ 0.5678 \end{pmatrix}$	$\begin{pmatrix} 1.2247 \\ -0.0000 \\ -0.0000 \\ 0.5000 \end{pmatrix}$	$\begin{pmatrix} 0.0536 \\ -0.2432 \\ -0.1238 \\ -0.0186 \end{pmatrix} \times 10^{-6}$	0.1098×10^{-6}
4	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1.2247 \\ -0.0000 \\ -0.0000 \\ 0.5000 \end{pmatrix}$	$\begin{pmatrix} 0.0725 \\ -0.1937 \\ -0.0964 \\ -0.035 \end{pmatrix} \times 10^{-6}$	0.0994×10^{-6}
5	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} -0.8769 \\ -0.2307 \\ -0.0516 \\ 0.7855 \end{pmatrix}$	$\begin{pmatrix} -0.8769 \\ -0.2307 \\ -0.0516 \\ 0.1820 \end{pmatrix}$	0.3353
6	$\begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1.0000 \\ 0.0000 \\ 3.0003 \\ -0.0001 \end{pmatrix}$	$\begin{pmatrix} -7.0171 \\ 9.5776 \\ 11.2059 \\ -52.3444 \end{pmatrix} \times 10^{-6}$	20.0362×10^{-6}
7	$\begin{pmatrix} 1 \\ 0 \\ 2 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1.0000 \\ -0.0000 \\ 3.0000 \\ -0.0000 \end{pmatrix}$	$\begin{pmatrix} -0.0886 \\ -0.1858 \\ 0.0402 \\ -0.7409 \end{pmatrix} \times 10^{-6}$	0.4831×10^{-6}

show that the proposed method is efficient and robust for solving the system nonsmooth equations.

Chapter 6.

Applications

In this chapter, we investigate some applications of the hybrid method proposed in Chapter 4.

6.1. Molecular conformation problem

6.1.1. Problem formulation

The minimum-energy configuration of a small cluster of atoms, molecules or ions is known as the *molecular conformation problem* [15]. It is a central problem in the study of cluster statics, or the topography of a potential energy function in an internal configuration space. From mathematical point of view, molecular conformation problem is a difficult global optimization problem which does not yield easily either to discrete or continuous optimization methods.

Suppose the cluster configuration of n atoms is given by the cartesian coordinates

$$X = \{X_i = (x_i^1, x_i^2, x_i^3) | i = 1, 2, \cdots, n\}.$$

The simplest assumption about interaction energy is that of two-body forces between component atoms, leading to the general two-body, n-atom potential

$$V_n(X) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} v(\| X_i - X_j \|), \tag{6.1}$$

where $\| X_i - X_j \|$ is the Euclidean distance between atoms X_i and X_j, i.e.,

$$\| X_i - X_j \| = \left(\sum_{k=1}^{3} (x_i^k - x_j^k)^2 \right)^{\frac{1}{2}} ;$$

$v : \mathbb{R}_+ \to \mathbb{R}$ is a pair potential function satisfying the conditions:

- $v(r) \to 0^-$ as $r \to \infty$,
- $v(r) \to \infty$ for $r < r_{\min}$ where $r_{\min} \geq 0$,
- $v'(r_0) = 0$ for some $r_{\min} \leq r_0 < \infty$
- $v''(r_0) > 0$ and $v(r_0) < 0$.

There are several different kinds of potential functions. However, we exclusively focus on the Lennard-Jones potential function. The Lennard-Jones potential (also referred to as the L-J potential, 6-12 potential or 12-6 potential) is a mathematically simple model that approximates the interaction between a pair of neutral atoms or molecules. It was first proposed in 1924 by John Lennard-Jones [48]. The most common expression of the L-J potential is

$$V_{lj} = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right], \tag{6.2}$$

where ε is the depth of the potential well, σ is the finite distance of which the inter-particle potential is zero and r is the distance between the particles. Another common expression of L-J potential is

$$V_{lj} = \varepsilon \left[\left(\frac{r_m}{r}\right)^{12} - 2\left(\frac{r_m}{r}\right)^{6} \right], \tag{6.3}$$

where $r_m = 2^{1/6}\sigma$ is the distance at which the potential reaches its minimum which is $-\varepsilon$. Parameters ε and σ can be fitted to reproduce experimental data or accurate quantum chemistry calculations. Due to its computational simplicity, the Lennard-Jones potential is used extensively in computer simulation even though more accurate potentials exist.

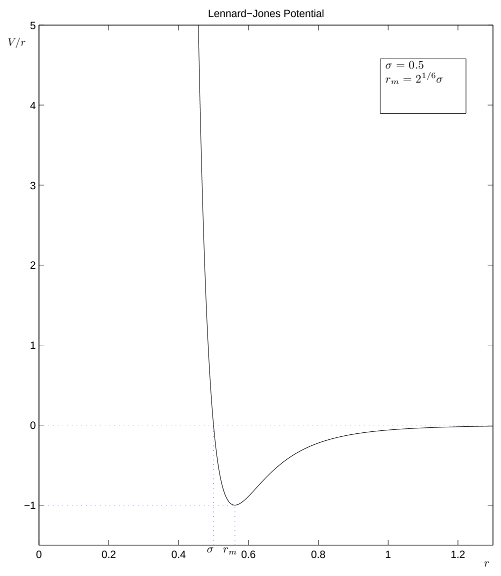


Figure 6.1.: Graph of strength versus distance for the 12-6 Lennard-Jones Potential.

Let $\varepsilon = 1$ and $r_m = 1$, and substitute $v(r)$ in Equation (6.1) by potential expression (6.3), the molecular conformation problem can be written as the following global optimization problem:

$$\begin{cases} \text{Minimize} & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left((\frac{1}{r_{ij}})^{12} - (\frac{1}{r_{ij}})^{6} \right) \\ \text{Subject to} & \\ & r_{ij} = \left(\sum_{k=1}^{3} (x_i^k - x_j^k)^2 \right)^{1/2} \\ & x_{\min}^k \le x_i^k \le x_{\max}^k \quad k = 1,2,3,\ i = 1,2,\cdots,n, \end{cases} \tag{6.4}$$

where $x_{\min}^k$ and $x_{\max}^k$, $k = 1,2,3$, are lower and upper bounds for the position of atoms.

6.1.2. Numerical results

In order to solve Problem (6.4), we use a single starting point $X^0 = \{X_1, \dots, X_n\}$ with $X_i = (x_i^1, x_i^2, x_i^3), x_i^k = 0.1(k + 3(i-1)), k = 1, 2, 3, i = 1, \dots, n.$ The numerical results obtained by the hybrid quasisecant method solving molecular conformation problem (6.4) are reported in Table 6.1. We present the results for the number of atoms from 2 to 23, i.e., the number of variables of Problem (6.4) is from 6 to 69 with a increment 3. From the table, we can see that all of the solutions obtained by the Algorithm HQSM are the same as the best known solutions. Better solutions are obtained for the number of atoms 18 and 22, new global minimizers of those two are as follows.

- 18 atoms: $x_{\min} =$

$$
\begin{array}{llllll}
(4.1517070 & 2.9404659 & 3.0691997 & 1.0125778 & 2.5982435 & 2.6209299 \\
1.6916219 & 3.2731873 & 1.9065144 & 2.7212357 & 2.8127015 & 1.9021200 \\
3.7016128 & 2.3271159 & 2.2109051 & 1.8090926 & 2.1470918 & 1.9511862 \\
2.7105479 & 1.9015668 & 2.5670481 & 3.5874697 & 1.9657620 & 3.2860929 \\
1.5002408 & 2.6673152 & 3.7092588 & 1.6952687 & 1.7867533 & 3.0233570 \\
2.5376094 & 2.2313789 & 3.6318135 & 3.4224619 & 2.8764679 & 3.9511865 \\
2.0547634 & 2.7896049 & 2.7958845 & 2.4420476 & 3.3602198 & 3.6456437 \\
3.0913141 & 2.8524167 & 2.9270360 & 3.4296522 & 3.8073075 & 3.2838784 \\
2.5585499 & 3.7290439 & 2.5482281 & 3.6077177 & 3.4629524 & 2.2061370)
\end{array}
$$

- 22 atoms: $x_{\min} =$

$$
\begin{array}{llllll}
(3.4951985 & 3.2748810 & 2.23156159 & 3.3009373 & 2.1936139 & 2.5717184 \\
2.7840535 & 4.8189627 & 3.1583095 & 1.7440650 & 3.1533782 & 2.8795260 \\
2.4569639 & 2.7860292 & 2.0432467 & 2.7969867 & 3.0235071 & 3.0225957 \\
2.2315421 & 2.1186534 & 2.9840040 & 3.1187181 & 2.2304580 & 3.7072315 \\
2.1670589 & 2.8171277 & 3.8953857 & 3.5281650 & 3.6044023 & 4.9095298 \\
2.3209326 & 3.8752850 & 3.5365904 & 3.1507374 & 3.2966110 & 3.9602726 \\
2.5091747 & 3.8827355 & 2.4265106 & 2.8989621 & 2.6319472 & 4.7592460 \\
3.3660739 & 3.9175613 & 3.1132804 & 3.9498275 & 4.7629169 & 3.4299800 \\
3.9611480 & 2.7040054 & 4.3350213 & 2.4085875 & 3.6611121 & 4.6419862 \\
3.1592928 & 4.3620844 & 4.1241997 & 4.3999931 & 3.8227839 & 2.8388174 \\
4.1185993 & 3.7707005 & 3.9345323 & 3.8750906 & 2.9171942 & 3.2293130)
\end{array}
$$

These results confirm that the proposed algorithm is efficient for solving molecular conformation problems.

6.2. Sensor localization problem

6.2.1. Problem formulation

A wireless sensor network (WSN) consist of spatially distributed autonomous sensor to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. and to cooperatively pass their data through the network to main location. The development of wireless sensor network was motivated by military application such as battlefield surveillance, now such networks are used in many industrial applications, such as industrial process, monitoring and control, machine health monitoring.

The wireless sensor network is built of “nodes” from a few to several hundred or even thousands, where each node is connected to one or several sensors. Wireless sensor network presents novel tradeoffs in system design. On the one hand, the low cost of nodes facilitates

Table 6.1.: Numerical results for Problem 6.4.

Number of atoms	Best known value	Best obtained value
2	-1.000000	-1.000000
3	-3.000000	-3.000000
4	-6.000000	-6.000000
5	-9.103852	-9.103852
6	-12.712062	-12.712062
7	-16.505384	-16.505384
8	-19.821489	-19.821489
9	-24.113360	-24.113360
10	-28.422532	-28.422531
11	-32.765970	-32.765970
12	-37.967600	-37.967600
13	-44.326801	-44.326801
14	-47.845157	-47.845156
15	-52.322627	-52.322627
16	-56.815742	-56.815741
17	-61.317995	-61.317994
18	-65.842309	-66.284568
19	-72.659782	-72.659782
20	-77.177043	-77.177042
21	-81.684571	-81.684571
22	-86.573675	-86.809782
23	-92.844461	-92.844461

massive scale and highly parallel computation. On the other hand, each node is likely to have limited power, limited reliability and only local communication with a modest number of neighbors. These application contexts and potential massive scale make it unrealistic to rely on careful placement or uniform arrangement of sensors. It is still impossible to localize each sensor by GPS because of the expensive cost and the limited power and memory of sensors. This leads to the area of *sensor localization problem* which intend to localize position of each sensor in a network by giving measured distances between the connected pairs of sensors. The distance information can be obtained by strategies like *time of arrival* (TOA), *time-difference of arrival* (TDoA) and *received signal strength* (RSS). Generally, there is some degree of error in the distance information because of the inaccuracy of measurement and power (or memory) constraints. If without any sensor's position beforehand, one can just estimate the relative position information of sensor network by giving measured distance of connected pairs of sensors. But for some networks, this is not good enough, such as networking for tracking, network for monitoring environmental information (temperature, sound levels, light, etc.). Thus, in order to obtain the absolute position of sensors, it should be assumed that we already know the position of a few *anchor nodes*. Those anchor nodes are called *beacon*.

For simplicity, let the sensors be placed on a plane. Suppose that we have m known points (anchors) which belong to $A = \{a_k \in \mathbb{R}^2 | k = 1, 2, \cdots, m\}$ and n unknown points (sensors) which belong to $S = \{x_i \in \mathbb{R}^2 | i = 1, 2, \cdots, n\}$. A collection of point-pair N_e is defined as

$$N_e = \{\{a_k, x_j\} | a_k \in A,\ x_j \in S\},$$

and a collection of point-pair N_u is defined as

$$N_u = \{\{x_i, x_j\} | x_i \in S,\ x_j \in S\}.$$

For a pair of points in N_e, we have a Euclidean distance measure d_{kj}. For a pair of points in N_u, we have a Euclidean distance measure d_{ij}. Then, the sensor localization problem is to find position of sensors (x_is), such that

$$\begin{aligned} \| a_k - x_j \|^2 &= d_{kj}^2 \qquad \text{for any } \{a_k, x_j\} \in N_e, \\ \| x_i - x_j \|^2 &= d_{ij}^2 \qquad \text{for any } \{x_i, x_j\} \in N_u. \end{aligned} \tag{6.5}$$

For a small number of sensors, it might be possible to compute sensor locations by solving the system of equations (6.5). Sturmfels [111] proposed a method for solving polynomial equations. However, solving polynomial system can be very expensive when there are a lot of sensors. Furthermore, this polynomial system may be inconsistent if the distances d_{kj} or d_{ij} have errors, which often occur in practice.

The polynomial system of equations (6.5) can also be convert into the following equivalent global optimization problem,

$$\min \left\{ \begin{aligned} e_1 \;&=\; \textstyle\sum_{\{a_k,x_j\}\in N_e} \left| \| a_k - x_j \|^2 - d_{kj}^2 \right| \\ &+\; \textstyle\sum_{\{x_i,x_j\}\in N_u,\ i<j} \left| \| x_i - x_j \|^2 - d_{ij}^2 \right| \end{aligned} \right\} \tag{6.6}$$

or

$$\min \left\{ \begin{aligned} e_2 \;&=\; \textstyle\sum_{\{a_k,x_j\}\in N_e} \left(\| a_k - x_j \|^2 - d_{kj}^2 \right)^2 \\ &+\; \textstyle\sum_{\{x_i,x_j\}\in N_u,\ i<j} \left(\| x_i - x_j \|^2 - d_{ij}^2 \right)^2 \end{aligned} \right\}. \tag{6.7}$$

Obviously, $x_1,\ x_2, \cdots, x_n$ are true sensor location if and only if the optimal value of problem (6.6) or problem (6.7) is zero. Thus, the sensor localization problem is equivalent to find the global minimizer of problem (6.6) or problem (6.7). Note that the problem (6.6) is nonsmooth problem, while the problem (6.7) is quadratic smooth problem, but both of them are nonconvex.

6.2.2. Numerical results

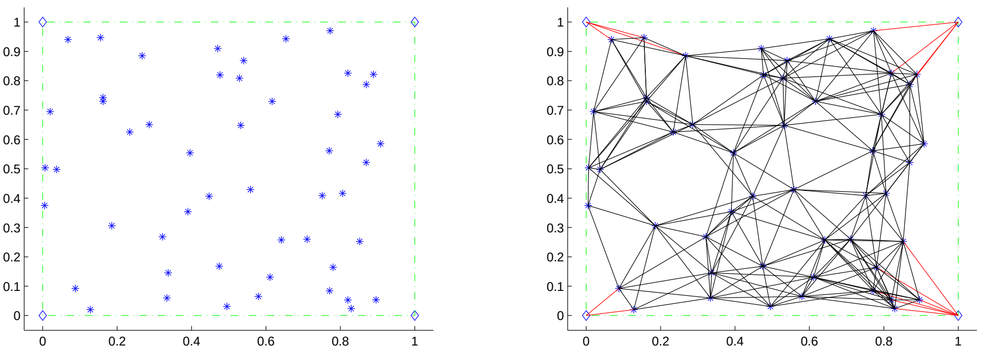
In this subsection, we solve some test sensor localization problems. The test problem generator is from SFSDP [59, 58, 60]. SFSDP is a MATLAB package for sensor localization problem using semidefinite programming (SDP). In SFSDP, there are four m-files, SFSDPplus.m, SFSDP.m, generateProblem.m and test_SFSDP.m. Among them, generateProblem.m is used to generate test problems. It is called by

[$xMatrix0, distanceMatrix0$] = generateProblem($sDim, noisyFac, radiorange$,...

$noOfSensors, anchorType, noOfAnchors, randSeed$).

The explanation of the input and output parameters can be found in the code. Figure 6.2.2 illustrates an example of sensor network generate by m-file "generateProblem.m" with 50

sensors and 4 anchors.



(a) Location of sensors (b) Connection of sensors

Figure 6.2.: An example of sensor network with 50 sensors and 4 anchors.

In order to evaluate the numerical performance of different solvers on sensor localization problems, Kim [59] developed the *root mean square distance*,

$$\text{RMSD} = \left(\frac{1}{N} \sum_{i=1}^{N} \|\hat{x}_i - x_i^*\|_2^2 \right),$$

where $\hat{x}_i$ is the true sensor location (for test problems, the true sensor location is known), x_i^* is the approximate sensor location obtained by optimization solvers. The root mean square distance reflects the average error between true sensor locations and obtained sensor locations. In order to evaluate the numerical performance of the hybrid quasisecant method for solving the molecular conformation problem, we apply measures such as average error and standard deviation of errors to investigate the statistical properties of the numerical results.

Following the notation of the root mean square distance, we consider $\hat{x}_i (i = 1, 2, \cdots, n)$ as the true sensor locations, $x_i^* (i = 1, 2, \cdots, n)$ as the approximation sensor locations obtained by optimization solver, and

$$d_i = \| \hat{x}_i - x_i^* \|, \; i = 1, 2, \cdots, n$$

as the error between the corresponding sensors. Then the following notations are used in evaluation of the algorithm.

- d_{ave}: the average error of corresponding sensors;
- d_{std}: the standard deviation of errors of corresponding sensors.

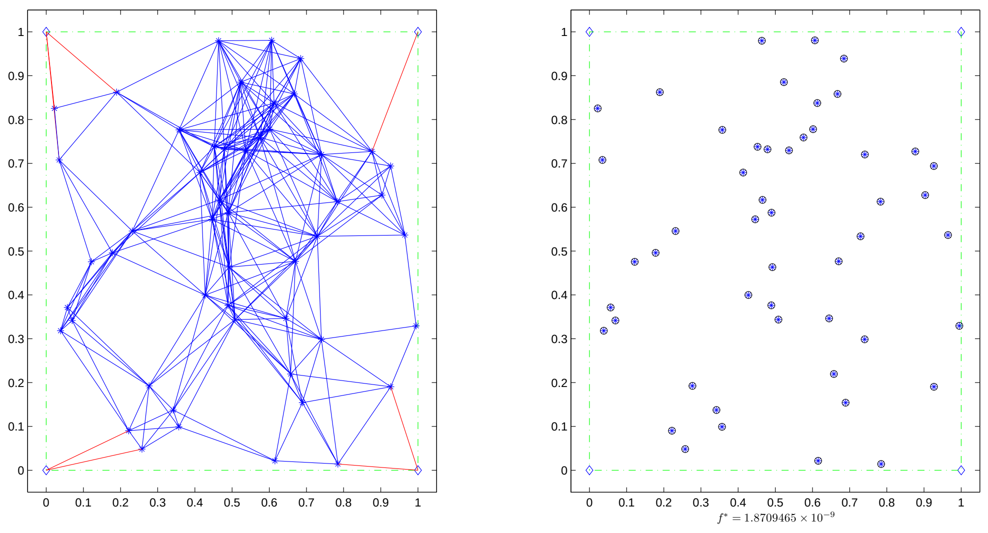
In the following, we solve some test problems of sensor localization by the hybrid quasisecant method proposed in the Chapter 4. In order to see the whole picture, we test sensor networks with the number of nodes 20-50 (with increment 10), 100-500 (with increment 100), 1000 and 2000 , respectively.

Table 6.2.: Numerical results for sensor localization problem.

No.	f^*	RMSD	d_{ave}	d_{std}
20	3.8751710×10^{-9}	$1.1409744 \times 10^{-11}$	3.0889847×10^{-5}	1.4022229×10^{-5}
30	2.0706732×10^{-9}	$9.5128657 \times 10^{-11}$	3.0889847×10^{-5}	1.4022229×10^{-5}
40	2.3013408×10^{-9}	$4.8699112 \times 10^{-10}$	1.7453671×10^{-5}	1.3676125×10^{-5}
50	1.8709465×10^{-9}	$1.1781585 \times 10^{-09}$	1.7453671×10^{-5}	1.3676125×10^{-5}
100	5.8333877×10^{-9}	$8.2989634 \times 10^{-10}$	1.7453671×10^{-5}	1.3676125×10^{-5}
200	2.4626913×10^{-9}	$2.6716611 \times 10^{-10}$	1.7453671×10^{-5}	1.3676125×10^{-5}
300	1.3537058×10^{-8}	$1.0683495 \times 10^{-10}$	1.7453671×10^{-5}	1.3676125×10^{-5}
400	1.3448554×10^{-8}	$1.1369356 \times 10^{-10}$	9.8926555×10^{-6}	3.9835417×10^{-6}
500	4.4267586×10^{-8}	$1.2075006 \times 10^{-09}$	3.4372143×10^{-5}	5.1096592×10^{-6}
1000	1.5312145×10^{-6}	$2.2130698 \times 10^{-08}$	1.4392109×10^{-4}	3.7667449×10^{-5}
2000	1.3000758×10^{-6}	$9.6764080 \times 10^{-09}$	9.6162666×10^{-5}	2.0721106×10^{-5}

The results are presented in Table 6.2. From the table, we can see that the hybrid quasisecant method solves all problems with high accuracy, however as the number of nodes increase the accuracy becomes slightly lower than that for small number of nodes. The root mean square distance of each problem are still extremely small, which implies that the hybrid quasisecant method is promising in solving the sensor localization problem with large number of nodes. The average and standard deviation of distance between obtained sensor localization and true sensor localization are very small, which yields the robustness of the hybrid quasisecant method.

Figures 6.3 and 6.4 demonstrate the results of sensor localization problems solved by the hybrid quasisecant method with 50 and 400 sensors, respectively. From Figures 6.3(b) and 6.4(b), we can see that all the true sensor locations (represented by “*”) are well estimated by obtained sensor locations (represented by “o”).



(a) Connection of sensors

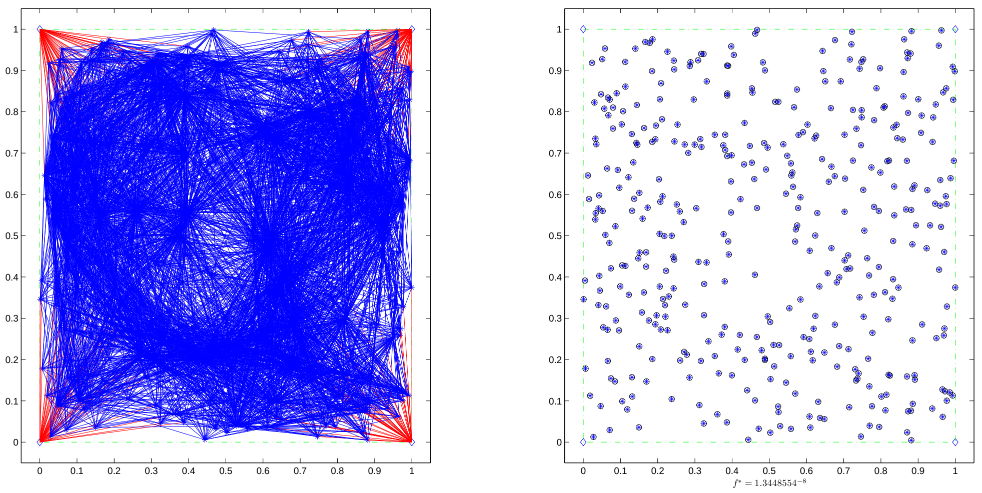
(b) Approximation of obtained sensor localization with the true sensor localization

Figure 6.3.: Results of sensor network with 50 sensors and 4 anchors.

Figure 6.5 depicts the decline of the objective function value when number of sensors equals 1000 and 2000. From Figure 6.5(a), the optimal solution is obtained after 20 iterations by the global search. From Figure 6.5(b), the optimal solution is obtained after 194 iterations by the global search. These two figures show that the global search strategy used in the hybrid quasisecant method is able to find good starting points (lower basins) for local search.

6.3. Conclusion

In this Chapter, we apply the hybrid quasisecant method proposed in Chapter 4 to solve some practical problems, more specifically, molecular conformation and sensor localization

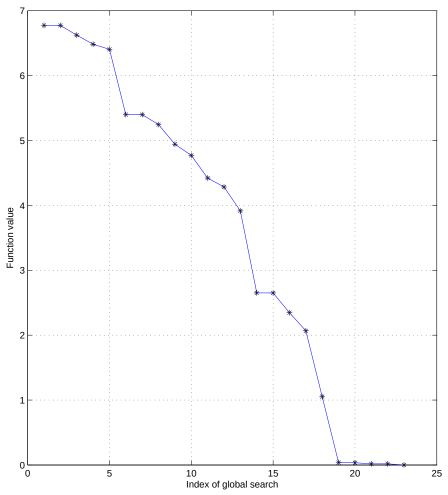


(a) Connection of sensors

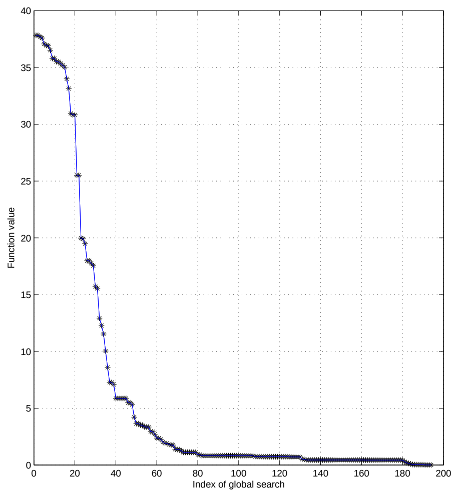
(b) Approximation of obtained sensor localization with the true sensor localization

Figure 6.4.: Results of sensor network with 400 sensors and 4 anchors.

problems. For the molecular conformation problem, the hybrid quasisecant method successfully solved all the test problems with number of atoms from 2 to 23. Better solutions were obtained for problems with number of atoms 18 and 22. For the sensor localization problem, the hybrid quasisecant method successfully solved all problems. In these sensor localization problems, the number of sensors is up to 2000, i.e., the number of variables is up to 4000. From the numerical performance of the hybrid quasisecant method for solving molecular conformation and sensor localization problems, we can conclude that the hybrid quasisecant method is efficient and robust for practical applications.



(a) Number os sensors is 1000



(b) Number of sensors is 2000

Figure 6.5.: Decrease of the objective function value

Chapter 7.

Conclusions and future work

7.1. Conclusions

This thesis presented numerical methods for solving nonsmooth and global optimization problems. The main contribution of this thesis are new numerical methods for solving global optimization problems and the system of nonsmooth equations.

First, we described and implemented the quasisecant method for solving nonsmooth optimization problem. The quasisecant method is similar to bundle-type methods and the main difference between these methods is that unlike the bundle methods the quasisecant uses quasisecants (which are approximate subgradients) to find search directions. The quasisecant method consist of outer iteration and inner iteration. The outer iteration adjusts a parameter which impacts the approximation of subdifferential. The inner iteration is employed to calculate the search direction. In each loop of the inner iteration, a quadratic programming problem is solved and the solution of quadratic programming problem either yield a descent direction or a new quasisecant which is used to improve the approximation of subdifferential. It was proved that the inner iteration can terminate in finite steps.

Using the quasisecant method, we developed a hybrid method for (nonsmooth) global optimization. This hybrid method consist of two phases: local search phase and global search phase. The local search phase is based upon the quasisecant method. It works as a solver for finding local minimizer. The global search phase is inspired by the inner iteration of the quasisecant method. It solves a series of subproblems which minimize the objective function over spheres with different radii to find better starting point for the local search. By solving the subproblems, the objective function is approximated by a series of piecewise linear function, and then for each piecewise linear approximation, a quadratic programming problem is considered. Thereafter, some points selected from the solutions of the subproblems are taken as starting points of the local search phase. The numerical experiments show that the proposed hybrid method is robust and efficient.

By combining the genetic algorithm and Hooke Jeeves method, we designed another hybrid method for solving constrained optimization problems. In order to overcome the slow convergence and problem with not a good accuracy of the genetic algorithm, we designed an acceleration operator based upon the Hooke Jeeves method. This operator can pick some suitable points from the generations of the genetic algorithm and do local search starting from these points. In this way, the acceleration operator improves the convergent rate and the accuracy of the genetic algorithm. We applied the proposed hybrid method to solve constrained optimization problems. Constraints in these problems were handled by the penalty function method. We considered two different penalty function models, quadratic penalty function and exact penalty function. The numerical performances shows that the results obtained by solving the exact penalty model is better than those obtained using the quadratic model, which demonstrates that the proposed hybrid method is good at solving nonsmooth optimization problems. We compared the numerical performance of the proposed hybrid method with other constrained optimization methods. The results shows that the proposed hybrid method is efficient and robust.

Based on the quasisecant method a numerical method is developed to solve the system of nonsmooth equations. The system of nonsmooth equations is first transformed into a nonsmooth optimization problem with a zero minimal objective function value. Then the quasisecant method is applied to solve the reformulated nonsmooth optimization problem. Numerical experiment show that this strategy allows one to solve the system of nonsmooth equations efficiently and robustly. The proposed method is applied to solve the systems of nonsmooth equations appeared in bilevel programming and nonlinear complementarity problems.

We applied the proposed hybrid method to solve the molecular conformation and sensor localization problems. For the molecular conformation problem, the hybrid method successfully solved all the problems with number of atoms from 2 to 23. Better solutions than those reported in the literature were obtained for problems with number of atoms 8 and 22. For the sensor localization problem, the hybrid method successfully solved all the problems. The number of sensors in these problems is up to 2000 resulting the number of variables up to 4000. These results demonstrate that the hybrid method developed in this thesis is highly accurate and efficient for solving the sensor locations problems with large number of sensors.

7.2. Future work

7.2.1. Application of global optimization

Nonsmooth and global optimization problems arise in many practical applications. In this thesis, the proposed hybrid method has been applied to solve the molecular conformation and sensor localization problems. Actually, there are many other areas about the application of global optimization techniques, such as acoustics equipment design, cancer therapy planning, chemical process modeling, data analysis, economic and financial forecasting, environment risk assessment and management, industrial product design, water management, and so on. In the future, we are planing to apply the proposed hybrid method in some of these areas.

7.2.2. Development of new hybrid methods

As we discussed in the literature review, there are three possible strategies to design a hybrid method: i) apply local search methods to improve the local search ability of global search methods; ii) apply global search methods to improve the global search ability of local search methods; iii) combine the global exploration properties of global search methods and the local exploitation properties of local search methods. The proposed hybrid methods in this thesis can be classified into the first group. We are planning to design hybrid methods based on the second and third strategies in the future.

Genetic algorithm is a widely accepted metaheuristic method for global optimization. But the numerical performance of genetic algorithm is deeply impacted by the phenomenon of premature convergence. For some problems, the population converge very fast but to a sub-optimal approximate solution. In order to avoid premature convergence but maintain the fast convergence rate, local search methods can be embedded into genetic algorithm to improve the performance of genetic algorithm. We can divide individuals in the population into two groups, global search group and local search group, with different properties. The individuals in global search group are used to explore new search area by applying crossover and mutation operators. The individuals in the local search group are employed as the starting point of local search methods. Once an individual in the global search group identified a new search area, it becomes an individual in local search group. On the other hand, if an individual in the local search group is exhausted to search a local area, it becomes an individual in global search group. In this way, the local search over the local search group guarantees the fast convergence, while the global search over the global search group maintains the diversity of the population all over the process, which in return, avoid the premature convergence phenomenon.

7.2.3. Parallel computation

Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently (in parallel). The primary objective of parallel computation is to solve large-scale problems more quickly.

One difficulty of parallel computation in numerical optimization is to explore the parallelism of algorithms. Some numerical optimization methods, like gradient-based methods, is hard to be parallelized in their current form because the information for the next iteration is supplied by the the previous iteration. These methods can only be partly parallelized (for example, parallelizes the evaluation of gradient by separately calculating each element on different processors) or fully parallelized (for example, implements the method on different processors with different starting points). On the other hand, some optimization methods, like metaheuristic methods, can be easily parallelized. For instance, in each iteration of genetic algorithm, the new offspring generated by crossover and mutation operators need to be evaluated by the fitness function, these jobs can be distributed among different processors.

Though the subject of parallel computation for optimization has been introduced for thirty years, there are only few mature numerical implementations. But the increasing size of optimization problems requires algorithms and implementations exclusively designed for parallel computation. This is going to be one of the directions of our future work.

Bibliography

[1] A.M. Bagirov. A method for minimization of quasidifferentiable functions. *Optimization Methods and Software*, 17(1):31–60, 2002.

[2] A.M. Bagirov and A.N. Ganjehlou. A quasisecant method for minimizing nonsmooth functions. *Optimization Methods & Software*, 25(1):3–18, 2010.

[3] A.M. Bagirov, M. Ghosh, and D. Webb. A derivative-free method for linearly constrained nonsmooth optimization. *Journal of Industrial and Management Optimization*, 2(3):319, 2006.

[4] A.M. Bagirov, L. Jin, N. Karmitsa, A.A. Nuaimat, and N. Sultanova. Subgradient method for nonconvex nonsmooth optimization. *Journal of Optimization Theory and Applications*, pages 1–20, 2012.

[5] A.M. Bagirov, B. Karasözen, and M. Sezer. Discrete gradient method: derivative-free method for nonsmooth optimization. *Journal of Optimization Theory and Applications*, 137(2):317–334, 2008.

[6] A.M. Bagirov and A.M. Rubinov. Cutting angle method and a local search. *Journal of Global Optimization*, 27:193–213, 2003.

[7] A.M. Bagirov and A.M. Rubinov. Local optimization method with global multidimensional search. *Journal of Global Optimization*, 32(2):161–179, 2005.

[8] J.R. Banga and W.D. Seider. Global optimization of chemical processes using stochastic algorithms. *Nonconvex Optimization and Its Applications*, 7:563–584, 1996.

[9] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear programming: theory and algorithms*. Wiley-interscience, 2006.

[10] S. Ben Hamida and M. Schoenauer. Aschea: New results using adaptive segregational constraint handling. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 884–889. IEEE, 2002.

[11] D.P. Bertsekas and S.K. Mitter. Steepest descent for optimization problems with nondifferentiable cost functionals. Technical report, DTIC Document, 1971.

[12] D.P. Bertsekas and S.K. Mitter. A descent numerical method for optimization problems with nondifferentiable cost functionals. *SIAM Journal on Control*, 11(4):637–652, 1973.

[13] M. Better, F. Glover, G. Kochenberger, and H. Wang. Simulation optimization: Applications in risk management. *International Journal of Information Technology & Decision Making*, 7(04):571–587, 2008.

[14] J.M. Bloemhof-Ruwaard, P. Van-Beek, L. Hordijk, and L.N. Van-Wassenhove. Interactions between operational research and environmental management. *European Journal of Operational Research*, 85(2):229–243, 1995.

[15] B.R. Brooks, R.E. Bruccoleri, B.D. Olafson, S. Swaminathan, M. Karplus, et al. Charmm: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4(2):187–217, 1983.

[16] D.G. Brooks and W.A. Verdini. Computational experience with generalized simulated annealing over continuous variables. *American Journal of Mathematical and Management Sciences*, 8(3-4):425–449, 1988.

[17] O. Buchtala, M. Klimek, and B. Sick. Evolutionary optimization of radial basis function classifiers for data mining applications. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 35(5):928–947, 2005.

[18] J.V. Burke, A.S. Lewis, and M.L. Overton. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM Journal on Optimization*, 15(3):751–779, 2005.

[19] Z.X. Cai and Y. Wang. A multiobjective optimization-based evolutionary algorithm for constrained optimization. *Evolutionary Computation, IEEE Transactions on*, 10(6):658–675, 2006.

[20] V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[21] B. Chen and P.T. Harker. Smooth approximations to nonlinear complementarity problems. *SIAM Journal on Optimization*, 7:403, 1997.

[22] C. Chen and O.L. Mangasarian. A class of smoothing functions for nonlinear and mixed complementarity problems. *Computational Optimization and Applications*, 5(2):97–138, 1996.

[23] Z.W. Chen, S.Q. Qiu, and Y.J. Jiao. A penalty-free method for equality constrained optimization. *Journal of Industrial and Management Optimization*, 9(2):391–409, 2013.

[24] E.W. Cheney and A.A. Goldstein. Newton's method for convex programming and tchebycheff approximation. *Numerische Mathematik*, 1(1):253–268, 1959.

[25] F.H. Clarke. *Optimization and nonsmooth analysis*, volume 5. Siam, 1990.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[27] V.F. Deḿanov. Smoothness of nonsmooth functions. In *Nonsmooth Optimization and Related Topics*, pages 79–88. Springer, 1989.

[28] C.A. Floudas. *Deterministic global optimization: theory, methods and applications*, volume 37. Springer, 2000.

[29] D.B. Fogel. *System identification through simulated evolution: A machine learning approach to modeling*. Ginn Press, 1991.

[30] D.B. Fogel. *Artificial intelligence through simulated evolution*. Wiley-IEEE Press, 2009.

[31] K.R. Fowler and C.T. Kelley. Pseudo-transient continuation for nonsmooth nonlinear equations. *SIAM Journal on Numerical Analysis*, pages 1385–1406, 2006.

[32] A. Frangioni. Solving semidefinite quadratic problems within nonsmooth optimization algorithms. *Computers & operations research*, 23(11):1099–1118, 1996.

[33] S.A. Gabriel and J.J. Moré. Smoothing of mixed complementarity problems. *Complementarity and Variational Problems: State of the Art*, pages 105–116, 1997.

[34] Y.L. Gao, H.G. Xue, and P.P. Shen. A new rectangle branch-and-reduce approach for solving nonconvex quadratic programming problems. *Applied Mathematics and Computation*, 168(2):1409–1418, 2005.

[35] R.P. Ge and Y.F. Qin. The globally convexized filled functions for global optimization. *Applied Mathematics and Computation*, 35(2):131–158, 1990.

[36] J.L. Goffin. On convergence rates of subgradient optimization methods. *Mathematical Programming*, 13(1):329–347, 1977.

[37] D. Goldberg. Genetic algorithms in optimization, search and machine learning. *Addison Wesley, New York. Eiben AE, Smith JE (2003) Introduction to Evolutionary Computing. Springer. Jacq J, Roux C (1995) Registration of non-segmented images using a genetic algorithm. Lecture notes in computer science*, 905:205–211, 1989.

[38] D.E. Goldberg and J.H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[39] D.E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1989.

[40] M. Haarala, K. Miettinen, and M.M. Mäkelä. New limited memory bundle method for large-scale nonsmooth optimization. *Optimization Methods and Software*, 19(6):673–692, 2004.

[41] N. Haarala, K. Miettinen, and M.M. Mäkelä. Globally convergent limited memory bundle method for large-scale nonsmooth optimization. *Mathematical Programming*, 109(1):181–205, 2007.

[42] A.R. Hedar and M. Fukushima. Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization. *Optimization Methods and Software*, 17(5):891–912, 2002.

[43] A.R. Hedar and M. Fukushima. Derivative-free filter simulated annealing method for constrained continuous global optimization. *Journal of Global Optimization*, 35(4):521–549, 2006.

[44] R. Hooke and T.A. Jeeves. “direct search”solution of numerical and statistical problems. *Journal of the ACM (JACM)*, 8(2):212–229, 1961.

[45] R. Horst. *Introduction to global optimization*. Springer, 2000.

[46] R. Horst, P.M. Pardalos, and H.E. Romeijn. *Handbook of global optimization*, volume 2. Springer, 2002.

[47] R. Horst and H. Tuy. *Global optimization: Deterministic approaches*. Springer, 1996.

[48] J.E. Jones. On the determination of molecular fields. ii. from the equation of state of a gas. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 106(738):463–477, 1924.

[49] N.H. Josephy. Newton's method for generalized equations. Technical report, DTIC Document, 1979.

[50] N.H. Josephy. Quasi-newton methods for generalized equations. Technical report, DTIC Document, 1979.

[51] A.R. Kan and G.T. Timmer. Stochastic global optimization methods part i: Clustering methods. *Mathematical Programming*, 39(1):27–56, 1987.

[52] C. Kanzow. Some noninterior continuation methods for linear complementarity problems. *SIAM Journal on Matrix Analysis and Applications*, 17:851, 1996.

[53] C. Kanzow and H. Jiang. A continuation method for (strongly) monotone variational inequalities. *Mathematical Programming*, 81(1):103–125, 1998.

[54] C. Kanzow and H.D. Qi. A qp-free constrained newton-type method for variational inequality problems. *Mathematical Programming*, 85(1):81–106, 1999.

[55] R. Karuppiah and I.E. Grossmann. Global optimization for the synthesis of integrated water systems in chemical processes. *Computers & Chemical Engineering*, 30(4):650–673, 2006.

[56] M. Kasper. Shape optimization by evolution strategy. *Magnetics, IEEE Transactions on*, 28(2):1556–1560, 1992.

[57] J.E. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial & Applied Mathematics*, 8(4):703–712, 1960.

[58] S.Y. Kim and M. Kojima. Semidefinite programming relaxations for sensor network localization. In *Computer-Aided Control System Design (CACSD), 2010 IEEE International Symposium on*, pages 19–23. IEEE, 2010.

[59] S.Y. Kim, M. Kojima, H. Waki, and M. Yamashita. Sfsdp: a sparse version of full semidefinite programming. 2009.

[60] S.Y. Kim, M. Kojima, H. Waki, and M. Yamashita. Algorithm 920: Sfsdp: A sparse version of full semidefinite programming relaxation for sensor network localization problems. *ACM Transactions on Mathematical Software (TOMS)*, 38(4):27, 2012.

[61] S. Kirkpatrick, C.D. Gelatt Jr, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[62] K.C. Kiwiel. *Methods of descent for nondifferentiable optimization*, volume 1133. Springer-Verlag Berlin, 1985.

[63] K.C. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming*, 46(1-3):105–122, 1990.

[64] K.C. Kiwiel. A tilted cutting plane proximal bundle method for convex nondifferentiable optimization. *Operations research letters*, 10(2):75–81, 1991.

[65] S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[66] V. Kvasnička and J. Pospíchal. A hybrid of simplex method and simulated annealing. *Chemometrics and Intelligent Laboratory Systems*, 39(2):161–173, 1997.

[67] C. Lemaréchal. Abstracts of ix international symposium on mathematical programming. In *Budapest, Hungary*.

[68] C. Lemaréchal. An extension of davidon methods to non-differentiable problems. In *Nondifferentiable optimization*, pages 95–109. Springer, 1975.

[69] C. Lemaréchal. Nondifferentiable optimisation subgradient and ε-subgradient methods. In *Optimization and Operations Research*, pages 191–199. Springer, 1976.

[70] C. Lemaréchal. Numerical experiments in nonsmooth optimization. 1982.

[71] C. Lemaréchal, J.J. Strodiot, and A. Bihain. On a bundle algorithm for nonsmooth optimization. *Nonlinear Programming*, 4(0), 1981.

[72] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168, 1944.

[73] A.V. Levy and A. Montalvo. The tunneling algorithm for the global minimization of functions. *SIAM Journal on Scientific and Statistical Computing*, 6(1):15–29, 1985.

[74] D.H. Li, N. Yamashita, and M. Fukushima. Nonsmooth equation based bfgs method for solving kkt systems in mathematical programming. *Journal of Optimization Theory and Applications*, 109(1):123–167, 2001.

[75] K.F. Lim, Beliakov G., and Batten L.M. Predicting molecular structures: Application of the cutting angle method. *Journal of Optimization Theory and Applications*, 5:3884–3890, 2003.

[76] M. Locatelli. Simulated annealing algorithms for continuous global optimization: convergence conditions. *Journal of Optimization Theory and applications*, 104(1):121–133, 2000.

[77] R. Lohmann. Application of evolution strategy in parallel populations. In *Parallel Problem Solving from Nature*, pages 198–208. Springer, 1991.

[78] L. Lukšan and J. Vlček. Globally convergent variable metric method for convex nonsmooth unconstrained minimization1. *Journal of Optimization Theory and Applications*, 102(3):593–613, 1999.

[79] L. Lukšan and J. Vlcek. Test problems for nonsmooth unconstrained and linearly constrained optimization. *Technická zpráva*, 798, 2000.

[80] D.W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441, 1963.

[81] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. E-quation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087, 1953.

[82] E. Mezura-Montes and C.A.C. Coello. A simple multimembered evolution strategy to solve constrained optimization problems. *Evolutionary Computation, IEEE Transactions on*, 9(1):1–17, 2005.

[83] J.A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[84] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

[85] J.V. Outrata, M. Kočvara, and J. Zowe. *Nonsmooth approach to optimization problems with equilibrium constraints: Theory, applications, and numerical results*, volume 28. Springer, 1998.

[86] A.J. Owens, M.J. Walsh, and L.J. Fogel. *Artificial intelligence through simulated evolution*. 1966.

[87] J.S. Pang. Newton's method for b-differentiable equations. *Mathematics of Operations Research*, pages 311–341, 1990.

[88] J.S. Pang and D. Chan. Iterative methods for variational and complementarity problems. *Mathematical Programming*, 24(1):284–313, 1982.

[89] J.D. Pintér. Global optimization in action. continuous and lipschitz optimization: Algorithms, implementation and applications, 1996.

[90] E. Polak, D.Q. Mayne, and Y. Wardi. On the extension of constrained optimization algorithms from differentiable to nondifferentiable problems. *SIAM Journal on Control and Optimization*, 21(2):179–203, 1983.

[91] F.A. Potra, L. Qi, and D. Sun. Secant methods for semismooth equations. *Numerische Mathematik*, 80(2):305–324, 1998.

[92] W.H. Press and S.A. Teukolsky. Simulated annealing optimization over continuous spaces. *Computers in Physics*, 5:426, 1991.

[93] K.V. Price. Differential evolution: a fast and simple numerical optimizer. In *Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American*, pages 524–527. IEEE, 1996.

[94] L. Qi. Trust region algorithms for solving nonsmooth equations. *SIAM Journal on Optimization*, 5:219, 1995.

[95] L. Qi and X. Chen. A globally convergent successive approximation method for severely nonsmooth equations. *SIAM Journal on Control and Optimization*, 33(2):402–418, 1995.

[96] L. Qi and D. Sun. A survey of some nonsmooth equations and smoothing newton methods. *Progress in Optimization*, 30:121–146, 1999.

[97] S.M. Robinson. Generalized equations and their solutions, part i: Basic theory. *Point-to-Set Maps and Mathematical Programming*, pages 128–141, 1979.

[98] S.M. Robinson. Strongly regular generalized equations. *Mathematics of Operations Research*, pages 43–62, 1980.

[99] R.T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898, 1976.

[100] R.T. Rockafellar. *Convex analysis*, volume 28. Princeton university press, 1997.

[101] T.P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *Evolutionary Computation, IEEE Transactions on*, 4(3):284–294, 2000.

[102] H.S. Ryoo and N.V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, 1996.

[103] N.V. Sahinidis. Global optimization and constraint satisfaction: The branch-and-reduce approach. In *Global Optimization and Constraint Satisfaction*, pages 1–16. Springer, 2003.

[104] W. Schirotzek. Nonasymptotic necessary conditions for nonsmooth infinite optimization problems. *Journal of Mathematical Analysis and Applications*, 118(2):535–546, 1986.

[105] H. Schramm and J. Zowe. A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results. *SIAM Journal on Optimization*, 2(1):121–152, 1992.

[106] A. Shapiro. On concepts of directional differentiability. *Journal of Optimization Theory and Applications*, 66(3):477–487, 1990.

[107] N.Z. Shor, K.C. Kiwiel, and A. Ruszcayński. *Minimization methods for non-differentiable functions*. Springer-Verlag New York, Inc., 1985.

[108] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[109] J.J. Strodiot, V.H. Nguyen, and N. Heukemes. ε-optimal solutions in nondifferentiable convex programming and some related questions. *Mathematical Programming*, 25(3):307–328, 1983.

[110] M. Studniarski. Sufficient conditions for the stability of local minimum points in nonsmooth optimization. *Optimization*, 20(1):27–35, 1989.

[111] B. Sturmfels. *Solving systems of polynomial equations*. Number 97. American Mathematical Soc., 2002.

[112] D. Sun and J. Han. Newton and quasi-newton methods for a class of nonsmooth equations and related problems. *SIAM Journal on Optimization*, 7(2):463–480, 1997.

[113] V.N. Tarasov and N.K. Popova. A modification of the cutting-plane method with accelerated convergence. In *Nondifferentiable Optimization: Motivations and Applications*, pages 284–290. Springer, 1985.

[114] B. Thomas, F. Hoffmeister, and H. Schwefel. A survey of evolution strategies. 1991.

[115] A. Torn and A. Zilinskas. *Global optimization*. Springer-Verlag New York, Inc., 1989.

[116] D.E. Varberg and A.W. Roberts. Convex functions. *New York-London*, 1973.

[117] M.P. Vecchi and S. Kirkpatrick. Global wiring by simulated annealing. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2(4):215–222, 1983.

[118] J. Vlček and L. Lukšan. Globally convergent variable metric method for nonconvex nondifferentiable unconstrained minimization. *Journal of Optimization Theory and Applications*, 111(2):407–430, 2001.

[119] D.J. Wales and H.A. Scheraga. Global optimization of clusters, crystals, and biomolecules. *Science*, 285(5432):1368–1372, 1999.

[120] Y. Wang, Z. Cai, Y. Zhou, and Z. Fan. Constrained optimization based on hybrid evolutionary algorithm and adaptive constraint-handling technique. *Structural and Multidisciplinary Optimization*, 37(4):395–413, 2009.

[121] D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994.

[122] P. Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. In *Nondifferentiable Optimization*, pages 145–173. Springer, 1975.

[123] P. Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11(1):128–149, 1976.

[124] Z.Y. Wu, H.W.J. Lee, L.S. Zhang, and X.M. Yang. A novel filled function method and quasi-filled function method for global optimization. *Computational Optimization and Applications*, 34(2):249–272, 2006.

[125] Z.Y. Wu, M. Mammadov, F.S. Bai, and Y.J. Yang. A filled function method for nonlinear equations. *Applied Mathematics and Computation*, 189(2):1196–1204, 2007.

[126] Z.Y. Wu, L.S. Zhang, K.L. Teo, and F.S. Bai. New modified function method for global optimization. *Journal of Optimization Theory and Applications*, 125(1):181–203, 2005.

[127] W. Xie and N.V. Sahinidis. A branch-and-reduce algorithm for the contact map overlap problem. In *Research in Computational Molecular Biology*, pages 516–529. Springer, 2006.

[128] H. Xu and B.M. Glover. New version of the newton method for nonsmooth equations. *Journal of Optimization Theory and Applications*, 93(2):395–415, 1997.

[129] X. Yao and Y. Liu. Fast evolutionary programming. In *Evolutionary Programming*, pages 451–460. Citeseer, 1996.

[130] X. Yao, Y. Liu, and G.M. Lin. Evolutionary programming made faster. *Evolutionary Computation, IEEE Transactions on*, 3(2):82–102, 1999.

[131] J. Yen, J.C. Liao, B. Lee, and R. David. A hybrid approach to modeling metabolic systems using a genetic algorithm and simplex method. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 28(2):173–191, 1998.

[132] K.F.C. Yiu, Y. Liu, and K.L. Teo. A hybrid descent method for global optimization. *Journal of Global Optimization*, 28(2):229–238, 2004.

[133] C.J. Yu, K.L. Teo, L.S. Zhang, and Y.Q. Bai. A new exact penalty function method for continuous inequality constrained optimization problems. *Journal of Industrial and Management Optimization*, 6(4):895, 2010.

[134] C. Zhang and H.P. Wang. Mixed-discrete nonlinear optimization with simulated annealing. *Engineering Optimization*, 21(4):277–291, 1993.

[135] J. Zowe and H. Schramm. Bundle trust methods: Fortran codes for nonsmooth optimization. user's guide. *Preprint*, 259, 2000.

Appendix A.

Test problems for minimax optimization

Problem 2.1 (CB2 Problem 2.1 [79])

$$\begin{aligned}
F(x) &= \max_{1\leq i\leq 3} f_i(x),\\
f_1(x) &= x_1^2 + x_2^4,\\
f_2(x) &= (2-x_1)^2 + (2-x_2)^2,\\
f_3(x) &= 2\exp(x_2 - x_1),\\
\bar{x}_1 &= 2, \quad \bar{x}_2 = 2.
\end{aligned}$$

Problem 2.2 (WF Problem 2.2 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 3} f_i(x),\\
f_1(x) &= \frac{1}{2}(x_1 + \frac{10x_1}{x_1+0.1} + 2x_2^2),\\
f_2(x) &= \frac{1}{2}(-x_1 + \frac{10x_1}{x_1+0.1} + 2x_2^2),\\
f_3(x) &= \frac{1}{2}(x_1 - \frac{10x_1}{x_1+0.1} + 2x_2^2),\\
\bar{x}_1 &= 3, \quad \bar{x}_2 = 1.
\end{aligned}$$

Problem 2.3 (SPIRAL Problem 2.3 [79])

$$\begin{aligned}
F(x) &= \max(f_1(x), f_2(x)),\\
f_1(x) &= \left(x_1 - \sqrt{x_1^2+x_2^2}\cos\sqrt{x_1^2+x_2^2}\right)^2 + 0.005(x_1^2+x_2^2),\\
f_2(x) &= \left(x_2 - \sqrt{x_1^2+x_2^2}\sin\sqrt{x_1^2+x_2^2}\right)^2 + 0.005(x_1^2+x_2^2),\\
\bar{x}_1 &= 1.41831, \quad \bar{x}_2 = -4.79462.
\end{aligned}$$

Problem 2.4 (EVD52 Problem 2.4 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 6} f_i(x),\\
f_1(x) &= x_1^2 + x_2^2 + x_3^2 - 1,\\
f_2(x) &= x_1^2 + x_2^2 + (x_3 - 2)^2,\\
f_3(x) &= x_1 + x_2 + x_3 - 1,\\
f_4(x) &= x_1 + x_2 - x_3 + 1,\\
f_5(x) &= 2x_1^3 + 6x_2^2 + 2(5x_3 - x_1 + 1)^2,\\
f_6(x) &= x_1^2 - 9x_3,\\
\bar{x}_i &= 1, \quad i = 1,2,3.
\end{aligned}$$

Problem 2.5 (Rosen-Suzuki Problem 2.5 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 4} f_i(x),\\
f_1(x) &= x_1^2+x_2^2+2x_3^2+x_4^2-5x_1-5x_2-21x_3+7x_4,\\
f_2(x) &= f_1(x)+10(x_1^2+x_2^2+x_3^2+x_4^2+x_1-x_2+x_3-x_4-8),\\
f_3(x) &= f_1(x)+10(x_1^2+2x_2^2+x_3^2+2x_4^2-x_1-x_4-10),\\
f_4(x) &= f_1(x)+10(2x_1^2+x_2^2+x_3^2+2x_1-x_2-x_4-5),\\
\bar{x}_i &= 0, \quad i=1,2,3,4.
\end{aligned}$$

Problem 2.6 (Polak 6 Problem 2.6 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 4} f_i(x),\\
f_1(x) &= (x_1-(x_4+1)^4)^2+(x_2-(x_1-(x_4+1)^4)^4)^2+2x_3^2\\
&\quad +x_4^2-5(x_1-(x_4+1)^4)-5(x_2-(x_1-(x_4+1)^4)^4)-21x_3+7x_4,\\
f_2(x) &= f_1(x)+10((x_1-(x_4+1)^4)^2+(x_2-(x_1-(x_4+1)^4)^4)^2\\
&\quad +x_3^2+x_4^2+(x_1-(x_4+1)^4)-(x_2-(x_1-(x_4+1)^4)^4)+x_3-x_4-8),\\
f_3(x) &= f_1(x)+10((x_1-(x_4+1)^4)^2+2(x_2-(x_1-(x_4+1)^4)^4)^2\\
&\quad +x_3^2+2x_4^2-(x_1-(x_4+1)^4)-x_4-10),\\
f_4(x) &= f_1(x)+10((x_1-(x_4+1)^4)^2+(x_2-(x_1-(x_4+1)^4)^4)^2\\
&\quad +x_3^2+2(x_1-(x_4+1)^4)-(x_2-(x_1-(x_4+1)^4)^4)-x_4-5),\\
\bar{x}_i &= 0, \quad i=1,2,3,4.
\end{aligned}$$

Problem 2.7 (PBC3 Problem 2.7 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 21} |f_i(x)|,\\
f_i(x) &= \frac{x_3}{x_2}\exp(-t_i x_1)\sin(t_i x_2) - y_i,\\
y_i &= \frac{3}{20}e^{-t_i} + \frac{1}{52}e^{-5t_i} - \frac{1}{65}e^{-2t_i}(3\sin 2t_i + 11\cos 2t_i),\\
t_i &= 10(i-1)/20, \quad 1 \le i \le 21,\\
\bar{x}_i &= 1, \quad 1 \le i \le 3.
\end{aligned}$$

Problem 2.8 (Kowalik-Osborne Problem 2.9 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 11} |f_i(x)|,\\
f_i(x) &= \frac{x_1(u_i^2 + x_2 u_i)}{u_i^2 + x_3 u_i + x_4} - y_i.
\end{aligned}$$

i	y_i	u_i	i	y_i	u_i
1	0.1957	4.0000	7	0.0456	0.0125
2	0.1947	2.0000	8	0.0342	0.1000
3	0.1735	1.0000	9	0.0323	0.0833
4	0.1600	0.5000	10	0.0235	0.0714
5	0.0844	0.2500	11	0.0246	0.0625
6	0.0627	0.1670			

$$\bar{x}_1 = 0.250, \ \ \bar{x}_2 = 0.390, \ \ \bar{x}_3 = 0.415, \ \ \bar{x}_4 = 0.390.$$

Problem 2.9 (Polak 2 Problem 2.22 [79])

$$\begin{aligned}
F(x) &= \max\{f(x+2e_2), f(x-2e_2)\},\\
f(x) &= \exp(10^{-8}x_1^2 + x_2^2 + x_3^2 + 4x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2 + x_9^2 + x_{10}^2),\\
e_2 &= \text{second column of the unit matrix},\\
\bar{x}_1 &= 100, \quad \bar{x}_i = 0.1, \quad 2 \le i \le 10.
\end{aligned}$$

Problem 2.10 (Polak 3 Problem 2.23 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 10} f_i(x),\\
f_i(x) &= \sum_{j=0}^{10} \frac{1}{i+j} \exp\left((x_{j+1} - \sin(i-1+2j))^2\right),\\
\bar{x}_i &= 1, \quad 1 \le i \le 11.
\end{aligned}$$

Problem 2.11 (Davidson 2 Problem 2.10 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 20} |f_i(x)|,\\
f_i(x) &= (x_1 + x_2 t_i - \exp(t_i))^2 + (x_3 + x_4\sin(t_i) - \cos(t_i))^2.\\
t_i &= 0.2i, \qquad 1 \le i \le 20,\\
\bar{x}_1 &= 25, \quad \bar{x}_2 = 5, \quad \bar{x}_3 = -5, \quad \bar{x}_4 = -1.
\end{aligned}$$

Problem 2.12 (OET5 Problem 2.11 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 21} |f_i(x)|, \\
f_i(x) &= x_4 - (x_1 t_i^2 + x_2 t_i + x_3)^2 - \sqrt{t_i}. \\
t_i &= 0.25 + 0.75(i-1)/20, \qquad 1 \le i \le 21, \\
\bar{x}_i &= 1.0, \quad 1 \le i \le 4.
\end{aligned}$$

Problem 2.13 (OET6 Problem 2.12 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 21} |f_i(x)|, \\
f_i(x) &= x_1 e^{x_3 t_i} + x_2 e^{x_4 t_i} - \frac{1}{1+t_i}, \\
t_i &= -0.5 + (i-1)/20, \qquad 1 \le i \le 21, \\
\bar{x}_1 &= 1.0, \quad \bar{x}_2 = 1.0, \quad \bar{x}_3 = -3.0, \quad \bar{x}_4 = -1.0.
\end{aligned}$$

Problem 2.14 (EXP Problem 2.14 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 21} f_i(x), \\
f_i(x) &= \frac{x_1 + x_2 t_i}{1 + x_3 t_i + x_4 t_i^2 + x_5 t_i^3} - \exp(t_i), \\
t_i &= -1 + (i-1)/10, \qquad 1 \le i \le 21, \\
\bar{x}_1 &= 0.5, \quad \bar{x}_2 = 0, \quad \bar{x}_3 = 0, \quad \bar{x}_4 = 0, \quad \bar{x}_5 = 0.
\end{aligned}$$

Problem 2.15 (PBC1 Problem 2.15 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 30} |f_i(x)|,\\
f_i(x) &= \frac{x_1 + x_2 t_i + x_3 t_i^2}{1 + x_4 t_i + x_5 t_i^2} - \frac{\sqrt{(8t_i - 1)^2 + 1}\,\arctan(8t_i)}{8t_i},\\
t_i &= -1 + 2(i-1)/29, \qquad 1 \le i \le 30,\\
\bar{x}_1 &= 0, \quad \bar{x}_2 = -1, \quad \bar{x}_3 = 10, \quad \bar{x}_4 = 1, \quad \bar{x}_5 = 10.
\end{aligned}$$

Problem 2.16 (EVD61 Problem 2.16 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 51} |f_i(x)|,\\
f_i(x) &= x_1 \exp(-x_2 t_i)\cos(x_3 t_i + x_4) + x_5 \exp(-x_6 t_i) - y_i,\\
y_i &= 0.5e^{-t_i} - e^{-2t_i} + 0.5e^{-3t_i} + 1.5e^{-1.5t_i}\sin 7t_i + e^{-2.5t_i}\sin 5t_i,\\
t_i &= 0.1(i-1), \qquad 1 \le i \le 51,\\
\bar{x}_1 &= 2, \quad \bar{x}_2 = 2, \quad \bar{x}_3 = 7,\\
\bar{x}_4 &= 0, \quad \bar{x}_5 = -2, \quad \bar{x}_6 = 1
\end{aligned}$$

Problem 2.17 (Filter Problem 2.18 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 41} |f_i(x)|,\\
f_i(x) &= \left(\frac{(x_1+(1+x_2)\cos\vartheta_i)^2+((1-x_2)\sin\vartheta_i)^2}{(x_3+(1+x_4)\cos\vartheta_i)^2+((1-x_4)\sin\vartheta_i)^2}\right)^{\frac{1}{2}}\cdot\\
&\quad \left(\frac{(x_5+(1+x_6)\cos\vartheta_i)^2+((1-x_6)\sin\vartheta_i)^2}{(x_7+(1+x_8)\cos\vartheta_i)^2+((1-x_8)\sin\vartheta_i)^2}\right)^{\frac{1}{2}} x_9 - y_i,\\
y_i &= |1-2t_i|, \quad \vartheta_i = \pi t_i\\
t_i &= 0.01(i-1), \quad 1\le i\le 6,\\
t_i &= 0.07+0.03(i-7), \quad 7\le i\le 20, \quad t_{21}=0.5,\\
t_i &= 0.54+0.03(i-22), \quad 22\le i\le 35,\\
t_i &= 0.95+0.01(i-36), \quad 36\le i\le 41,\\
\bar{x}_1 &= 0.00, \quad \bar{x}_2 = 1.00, \quad \bar{x}_3 = 0.00, \quad \bar{x}_4 = -0.15,\\
\bar{x}_5 &= 0.00, \quad \bar{x}_6 = -0.68, \quad \bar{x}_7 = 0.00, \quad \bar{x}_8 = -0.72,\\
\bar{x}_9 &= 0.37.
\end{aligned}$$

Problem 2.18 (Wong 1 Problem 2.19 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 5} f_i(x),\\
f_1(x) &= (x_1-10)^2+5(x_2-12)^2+x_3^4+3(x_4-11)^2+10x_5^6+7x_6^2+x_7^4\\
&\quad -4x_6x_7-10x_6-8x_7,\\
f_2(x) &= f_1(x)+10(2x_1^2+3x_2^4+x_3+4x_4^2+5x_5-127),\\
f_3(x) &= f_1(x)+10(7x_1+3x_2+10x_3^2+x_4-x_5-282),\\
f_4(x) &= f_1(x)+10(23x_1+x_2^2+6x_6^2-8x_7-196),\\
f_5(x) &= f_1(x)+10(4x_1^2+x_2^2-3x_1x_2+2x_3^2+5x_6-11x_7),\\
\bar{x}_1 &= 1, \quad \bar{x}_2=2, \quad \bar{x}_3=0, \quad \bar{x}_4=4, \quad \bar{x}_5=0, \quad \bar{x}_6=1, \quad \bar{x}_7=1.
\end{aligned}$$

Problem 2.19 (Wong 2 Problem 2.20 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 9} f_i(x),\\
f_1(x) &= x_1^2+x_2^2+x_1x_2-14x_1-16x_2+(x_3-10)^2+4(x_4-5)^2+(x_5-3)^2+\\
&\quad 2(x_6-1)^2+5x_7^2+7(x_8-11)^2+2(x_9-10)^2+(x_{10}-7)^2+45,\\
f_2(x) &= f_1(x)+10(3(x_1-2)^2+4(x_2-3)^2+2x_3^2-7x_4-120),\\
f_3(x) &= f_1(x)+10(5x_1^2+8x_2+(x_3-6)^2-2x_4-40),\\
f_4(x) &= f_1(x)+10(0.5(x_1-8)^2+2(x_2-4)^2+3x_5^2-x_6-30),\\
f_5(x) &= f_1(x)+10(x_1^2+2(x_2-2)^2-2x_1x_2+14x_5-6x_6),\\
f_6(x) &= f_1(x)+10(4x_1+5x_2-3x_7+9x_8-105),\\
f_7(x) &= f_1(x)+10(10x_1-8x_2-17x_7+2x_8),\\
f_8(x) &= f_1(x)+10(-3x_1+6x_2+12(x_9-8)^2-7x_{10}),\\
f_9(x) &= f_1(x)+10(-8x_1+2x_2+5x_9-2x_{10}-12),\\
\bar{x}_1 &= 2,\ \ \bar{x}_2=3,\ \ \bar{x}_3=5,\ \ \bar{x}_4=5,\ \ \bar{x}_5=1,\ \ \bar{x}_6=2,\\
\bar{x}_7 &= 7,\ \ \bar{x}_8=3,\ \ \bar{x}_9=6,\ \ \bar{x}_{10}=10.
\end{aligned}$$

Problem 2.20 (Wong 3 Problem 2.21 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 18} f_i(x),\\
f_1(x) &= x_1^2+x_2^2+x_1x_2-14x_1-16x_2+(x_3-10)^2+4(x_4-5)^2+(x_5-3)^2\\
&\quad +2(x_6-1)^2+5x_7^2+7(x_8-11)^2+2(x_9-10)^2+(x_{10}-7)^2+(x_{11}-9)^2\\
&\quad +10(x_{12}-1)^2+5(x_{13}-7)^2+4(x_{14}-14)^2+27(x_{15}-1)^2+x_{16}^4+(x_{17}-2)^2\\
&\quad +13(x_{18}-2)^2+(x_{19}-3)^2+x_{20}^2+95,
\end{aligned}$$

$$
\begin{aligned}
f_2(x) &= f_1(x) + 10(3(x_1-2)^2 + 4(x_2-3)^2 + 2x_3^2 - 7x_4 - 120), \\
f_3(x) &= f_1(x) + 10(5x_1^2 + 8x_2 + (x_3-6)^2 - 2x_4 - 40), \\
f_4(x) &= f_1(x) + 10(0.5(x_1-8)^2 + 2(x_2-4)^2 + 3x_5^2 - x_6 - 30), \\
f_5(x) &= f_1(x) + 10(x_1^2 + 2(x_2-2)^2 - 2x_1x_2 + 14x_5 - 6x_6), \\
f_6(x) &= f_1(x) + 10(4x_1 + 5x_2 - 3x_7 + 9x_8 - 105), \\
f_7(x) &= f_1(x) + 10(10x_1 - 8x_2 - 17x_7 + 2x_8), \\
f_8(x) &= f_1(x) + 10(-3x_1 + 6x_2 + 12(x_9-8)^2 - 7x_{10}), \\
f_9(x) &= f_1(x) + 10(-8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12), \\
f_{10}(x) &= f_1(x) + 10(x_1 + x_2 + 4x_{11} - 21x_{12}), \\
f_{11}(x) &= f_1(x) + 10(x_1^2 + 5x_{11} - 8x_{12} - 28), \\
f_{12}(x) &= f_1(x) + 10(4x_1 + 9x_2 + 5x_{13}^2 - 9x_{14} - 87), \\
f_{13}(x) &= f_1(x) + 10(3x_1 + 4x_2 + 3(x_{13}-6)^2 - 14x_{14} - 10), \\
f_{14}(x) &= f_1(x) + 10(14x_1^2 + 35x_{15} - 79x_{16} - 92), \\
f_{15}(x) &= f_1(x) + 10(15x_2^2 + 11x_{15} - 61x_{16} - 54), \\
f_{16}(x) &= f_1(x) + 10(5x_1^2 + 2x_2 + 9x_{17}^4 - x_{18} - 68), \\
f_{17}(x) &= f_1(x) + 10(x_1^2 - x_2 + 19x_{19} - 20x_{20} + 19), \\
f_{18}(x) &= f_1(x) + 10(7x_1^2 + 5x_2^2 + x_{19}^2 - 30x_{20}), \\
\bar{x}_1 &= 2,\ \ \bar{x}_2 = 3,\ \ \bar{x}_3 = 5,\ \ \bar{x}_4 = 5,\ \ \bar{x}_5 = 1,\ \ \bar{x}_6 = 2,\ \ \bar{x}_7 = 7, \\
\bar{x}_8 &= 3,\ \ \bar{x}_9 = 6,\ \ \bar{x}_{10} = 10,\ \ \bar{x}_{11} = 2,\ \ \bar{x}_{12} = 2,\ \ \bar{x}_{13} = 6,\ \ \bar{x}_{14} = 15, \\
\bar{x}_{15} &= 1,\ \ \bar{x}_{16} = 2,\ \ \bar{x}_{17} = 1,\ \ \bar{x}_{18} = 2,\ \ \bar{x}_{19} = 1,\ \ \bar{x}_{20} = 3.
\end{aligned}
$$

Problem 2.21 (Watson Problem 2.24 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 31} |f_i(x)|, \\
f_1(x) &= x_1, \\
f_2(x) &= x_2 - x_1^2 - 1, \\
f_i(x) &= \sum_{j=2}^{n}(j-1)x_j\left(\frac{i-2}{29}\right)^{j-2} - \left[\sum_{j=1}^{n} x_j\left(\frac{i-2}{29}\right)^{j-1}\right]^2, \quad 3 \le j \le 31 \\
\bar{x}_i &= 0, \quad 1 \le i \le 20.
\end{aligned}$$

Problem 2.22 (Osborne 2 Problem 2.25 [79])

$$\begin{aligned}
F(x) &= \max_{1\le i\le 65} |f_i(x)|, \\
f_i(x) &= y_i - x_1\exp(-x_5 t_i) - x_2\exp(-x_6(t_i - x_9)^2) - x_3\exp(-x_7(t_i - x_{10})^2) - \\
&\quad x_4\exp(-x_8(t_i - x_{11})^2), \\
t_i &= 0.1(i-1), \quad 1 \le i \le 65. \\
\bar{x}_1 &= 1.30, \;\; \bar{x}_2 = 0.65, \;\; \bar{x}_3 = 0.65, \;\; \bar{x}_4 = 0.70, \;\; \bar{x}_5 = 0.60, \;\; \bar{x}_6 = 3.00, \\
\bar{x}_7 &= 5.00, \;\; \bar{x}_8 = 7.00, \;\; \bar{x}_9 = 2.00, \;\; \bar{x}_{10} = 4.50, \;\; \bar{x}_{11} = 5.50.
\end{aligned}$$

i	y_i	i	y_i	i	y_i	i	y_i	i	y_i
1	1.366	14	0.655	27	0.612	40	0.429	53	0.597
2	1.191	15	0.616	28	0.558	41	0.523	54	0.625
3	1.112	16	0.606	29	0.533	42	0.562	55	0.739
4	1.013	17	0.602	30	0.495	43	0.607	56	0.710
5	0.991	18	0.626	31	0.500	44	0.653	57	0.729
6	0.885	19	0.651	32	0.423	45	0.672	58	0.720
7	0.831	20	0.724	33	0.395	46	0.708	59	0.636
8	0.847	21	0.649	34	0.375	47	0.633	60	0.581
9	0.786	22	0.649	35	0.372	48	0.668	61	0.428
10	0.725	23	0.694	36	0.391	49	0.645	62	0.292
11	0.746	24	0.644	37	0.396	50	0.632	63	0.162
12	0.679	25	0.624	38	0.405	51	0.591	64	0.098
13	0.608	26	0.661	39	0.428	52	0.559	65	0.054

Problem 2.23 (Problem 1 [1])

$$\begin{aligned}
f(x) &= \max\{f_i(x):\ i=1,2,3\}+\min\{f_i(x):\ i=4,5,6\},\\
f_1(x) &= x_1^4+x_2^2,\\
f_2(x) &= (2-x_1)^2+(2-x_2)^2,\\
f_3(x) &= 2e^{-x_1+x_2},\\
f_4(x) &= x_1^2-2x_1+x_2^2-4x_2+4,\\
f_5(x) &= 2x_1^2-5x_1+x_2^2-2x_2+4,\\
f_6(x) &= x_1^2+2x_2^2-4x_2+1,\\
& \quad x\in\mathbb{R}^2,\quad x^0=(2,2),\quad x^*=(1,1),\quad f^*=2.
\end{aligned}$$

Problem 2.24 (Problem 2 [1])

$$f(x) = |x_1 - 1| + 100|x_2 - |x_1||$$

$$x \in \mathbb{R}^2, x^0 \in (-1.2, 1), x^* = (1, 1), f^* = 0$$

The function f can be represented as the difference of two convex function:

$$f(x) = f_1(x) - f_2(x),$$

where

$$f_1(x) = |x_1 - 1| + 200 \max\{0,\ |x_1| - x_2\}, \quad f_2(x) = 100(|x_1| - x_2).$$

Problem 2.25 (Problem 3 [1])

$$\begin{aligned} F(x) &= |x_1 - 1| + 100|x_2 - |x_1|| + 90|x_4 - |x_3|| + |x_3 - 1| \\ &\quad +10.1(|x_2 - 1| + |x_4 - 1|) + 4.95(|x_2 + x_4 - 2| - |x_2 - x_4|), \\ &\quad x \in \mathbb{R}^4,\ x^0 = (1, 3, 3, 1),\ x^* = (1, 1, 1, 1),\ f^* = 0. \end{aligned}$$

The function f can be represented as the difference of two convex functions:

$$f(x) = f_1(x) - f_2(x), \text{ where}$$

$$\begin{aligned} f_1(x) &= |x_1 - 1| + 200 \max\{0, |x_1| - x_2\} + 180 \max\{0, |x_3| - x_4\} \\ &\quad +|x_3 - 1| + 10.1(|x_2 - 1| + |x_4 - 1|) + 4.95|x_2 + x_4 - 2|, \\ f_2(x) &= 100(|x_1| - x_2) + 90(|x_3| - x_4) + 4.95|x_2 - x_4|. \end{aligned}$$

Appendix B.

Test problems for global optimization

Problem 3.1. Ackley function

- Numbers of variables: n variables.
- Definition:

$$f(x) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{-\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}.$$

- Box constraint: $-1.5 \leq x_i \leq 30,\ i = 1, 2, \cdots, n.$
- The global minimum: $x^* = (0, 0, \cdots, 0)$ with $f(x^*) = 0.$

Problem 3.2. Hump function

- Numbers of variables: $n = 2.$
- Definition:

$$f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

- Box constraint: $-5 \leq x_i \leq 5,\ i = 1, 2.$
- The global minimum: $x^* = (0.0898, -0.7126),\ (-0.0898, 0.7126)$ with $f(x^*) = -1.0316285.$

Problem 3.3. Griewank function

- Numbers of variables: n variables.
- Definition:

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$$

- Box constraint: $-600 \leq x_i \leq 600,\ i = 1, 2, \cdots, n.$
- The global minimum: $x^* = (0, 0, \cdots, 0)$ with $f(x^*) = 0.$

Problem 3.4. Rastrigin function

- Numbers of variables: n variables.
- Definition:

$$f(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i))$$

- Box constraint: $-5.12 \leq x_i \leq 5.12,\ i = 1, 2, \cdots, n.$
- The global minimum: $x^* = (0, 0, \cdots, 0)$ with $f(x^*) = 0.$

Problem 3.5. Rosenbrock function

- Numbers of variables: n variables.
- Definition:

$$f(x) = \sum_{i=1}^{n-1} \left[100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right].$$

- Box constraint: $-5 \leq x_i \leq 10,\ i = 1, 2, \cdots, n.$
- The global minimum: $x^* = (1, 1, \cdots, 1)$ with $f(x^*) = 0.$

Problem 3.6. Schwefel function

- Numbers of variables: n variables.

- Definition:
$$f(x) = 418.9829n - \sum_{n=1}^{n}(x \sin \sqrt{|x_i|})$$
- Box constraint: $-500 \leq x_i \leq 500,\ i = 1, 2, \cdots, n.$
- The global minimum: $x^* = 420.96914 \times (1, 1, \cdots, 1)$ with $f(x^*) = 1 \times 10^{-5}$.

Problem 3.7. Dixon $\&$ Price function

- Numbers of variables: n variables.
- Definition:
$$f(x) = (x_1 - 1)^2 + \sum_{i=1}^{n} i(2x_i^2 - x_{i-1})^2.$$
- Box constraint: $-10 \leq x_i \leq 10,\ i = 1, 2 \cdots, n.$
- The global minimum: $f(x^*) = 0.$

Problem 3.8. Levy function

- Numbers of variables: n variables.
- Definition:
$$\begin{aligned} f(x) &= \sin^2(\pi y_1) + \sum_{i=1}^{n-1} \left[(y_i - 1)^2(1 + 10\sin^2(\pi y_i + 1))\right] \\ &\quad +(y_n - 1)^2(1 + 10\sin^2(2\pi y_n)), \end{aligned}$$
where $y_i = 1 + \frac{x_i - 1}{4},\ i = 1, 2, \cdots, n.$
- Box constraint: $-10 \leq x_i \leq 10,\ i = 1, 2, \cdots, n.$
- The global minimum: $x^* = (1, 1, \cdots, 1)$ with $f(x^*) = 0.$

Problem 3.9. Michalewics function

- Numbers of variables: n variables.

- Definition:

$$f(x) = -\sum_{i=1}^{2} \sin(x_i)(\sin(ix_i^2/\pi))^{2m},$$

  where $m = 10$.

- Box constraint: $0 \le x_i \le \pi,\ i = 1, 2, \cdots, n.$

- The global minimum:

$$\begin{aligned} f(x^*) &= -1.8013 && \text{when } n = 2, \\ f(x^*) &= -4.687658 && \text{when } n = 5, \\ f(x^*) &= -9.66015 && \text{when } n = 10. \end{aligned}$$

Problem 3.10. Perm function I

- Numbers of variables: n variables.

- Definition:

$$f(x) = \sum_{i=1}^{n} \left[\sum_{j=1}^{n} (j^i + 0.5)((x_j/j)^i - 1) \right]^2$$

- Box constraint: $-n \le x_i \le n,\ i = 1, 2, \cdots, n.$

- The global minimum: $x^* = (1, 2, \cdots, n)$ with $f(x^*) = 0$.

Problem 3.11. Trid function

- Numbers of variables: n variables.

- Definition:

$$f(x) = \sum_{i=1}^{n} (x_i - 1)^2 - \sum_{i=2}^{n} x_i x_{i-1}$$

- Box constraint: $-n^2 \le x_i \le n^2,\ i = 1, 2, \cdots, n.$

- The global minimum:

$$\begin{aligned} f(x^*) &= -2 && \text{when } n = 2, \\ f(x^*) &= -30 && \text{when } n = 5, \\ f(x^*) &= -50 && \text{when } n = 6, \\ f(x^*) &= -210 && \text{when } n = 10. \end{aligned}$$

Problem 3.12. Zakharov function

- Numbers of variables: n variables.
- Definition:

$$f(x) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5 i x_i \right)^2 + \left(\sum_{i=1}^{n} 0.5 i x_i \right)^4$$

- Box constraint: $-5 \leq x_i \leq 10,\ i = 1, 2, \cdots, n.$
- The global minimum: $x^* = (0, 0, \cdots, 0)$ with $f(x^*) = 0$.

Problem 3.13. Beale function

- Numbers of variables: $n = 2$.
- Definition:

$$f(x) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2.$$

- Box constraint: $-4.5 \leq x_i \leq 4.5,\ i = 1, 2.$
- The global minimum: $x^* = (3, 0.5)$ with $f(x^*) = 0$.

Problem 3.14. Bohachevsky function I

- Numbers of variables: $n = 2$.

- Definition:

$$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7.$$

- Box constraint: $-100 \leq x_i \leq 100,\ i = 1, 2$.
- The global minimum: $x^* = (0, 0)$ with $f(x^*) = 0$.

Problem 3.15. Bohachevsky function II

- Numbers of variables: $n = 2$.
- Definition:

$$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3.$$

- Box constraint: $-100 \leq x_i \leq 100,\ i = 1, 2$.
- The global minimum: $x^* = (0, 0)$ with $f(x^*) = 0$.

Problem 3.16. Bohachevsky function III

- Numbers of variables: $n = 2$.
- Definition:

$$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3.$$

- Box constraint: $-100 \leq x_i \leq 100,\ i = 1, 2$.
- The global minimum: $x^* = (0, 0)$ with $f(x^*) = 0$.

Problem 3.17. Brainin function

- Numbers of variables: $n = 2$.

- Definition:

$$f(x) = (x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos(x_1) + 10.$$

- Box constraint: $-5 \leq x_1 \leq 10,\ 0 \leq x_2 \leq 15.$
- The global minimum: $x^* = (-\pi, 12.275),\ (\pi, 2.275),\ (9.42748, 2.475)$ with $f(x^*) = 0.397887.$

Problem 3.18. Colville function

- Numbers of variables: $n = 4.$
- Definition:

$$\begin{aligned} f(x) &= 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 \\ &\quad +10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.6(x_2 - 1)(x_4 - 1). \end{aligned}$$

- Box constraint: $-10 \leq x_i \leq 10,\ i = 1, 2, 3, 4.$
- The global minimum: $x^* = (1, 1, 1, 1)$ with $f(x^*) = 0.$

Problem 3.19. Matyas function

- Numbers of variables: $n = 2.$
- Definition:

$$f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2.$$

- Box constraint: $-10 \leq x_i \leq 10,\ i = 1, 2.$
- The global minimum: $x^* = (0, 0)$ with $f(x^*) = 0.$

Problem 3.20. Shubert function

- Numbers of variables: $n = 2.$

- Definition:

$$f(x) = \left(\sum_{i=1}^{5} i \cos((i+1)x_i + i) \right) \left(\sum_{i=1}^{5} i \cos((i+1)x_2 + i) \right).$$

- Box constraint: $-10 \leq x_i \leq 10, \ i = 1, 2.$

- The global minimum: 18 global minima with $f(x^*) = -186.7309$.

Appendix C.

Test problems for constrained optimization

Problem 4.1

- Numbers of variables: 13 variables.
- Definition:

  Objective function

$$f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

Inequality constraints

$$\begin{aligned}
g_1(x) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\
g_2(x) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\
g_3(x) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\
g_4(x) &= -8x_1 + x_{10} \leq 0 \\
g_5(x) &= -8x_2 + x_{11} \leq 0 \\
g_6(x) &= -8x_3 + x_{112} \leq 0
\end{aligned}$$

$$\begin{aligned}
g_7(x) &= -2x_4 - x_5 + x_{10} \leq 0 \\
g_8(x) &= -2x_6 - x_7 + x_{11} \leq 0 \\
g_9(x) &= -2x_8 - x_9 + x_{12} \leq 0
\end{aligned}$$

- Box constraint: $0 \leq x_i \leq u_i,\ i = 1, 2, \cdots, 13$, and $u = (1, 1, \cdots, 1, 100, 100, 100, 1)$.
- The global minimum: $x^* = (1, 1, \cdots, 1, 3, 3, 3, 1)$ with $f(x^*) = -15$.

Problem 4.2

- Numbers of variables: n variables.
- Definition:

Objective function

$$f(x) = -\left| \frac{\sum_{i=1}^{n} cos^4(x_i) - 2\prod_{i=1}^{n} cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

Inequality constraints

$$\begin{aligned}
g_1(x) &= -\prod_{i=1}^{n} x_i + 0.75 \leq 0 \\
g_2(x) &= \sum_{i=1}^{n} x_i - 7.5n \leq 0
\end{aligned}$$

- Box constraint: $0 \leq x_i \leq 10,\ i = 1, 2, \cdots, n.$
- The global minimum: at $n = 20$, $f(x^*) = -0.803619.$

Problem 4.3

- Numbers of variables: n variables.
- Definition:

  Objective function

$$f(x) = -(\sqrt{n})^n \prod_{i=1}^{n} x_i$$

  Equality constraints

$$h_1(x) \;\; = \;\; \sum_{i=1}^{n} x_i^2 - 1 = 0$$

- Box constraint: $0 \leq x_i \leq 1,\ i = 1, 2, \cdots, n.$
- The global minimum: $x^* = (1/n^{0.5}, 1/n^{0.5}, \cdots, 1/n^{0.5})$ with $f(x^*) = -1.$

Problem 4.4

- Numbers of variables: 5 variables.
- Definition:

  Objective function

$$f(x) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

Inequality constraints

$$\begin{aligned}
g_1(x) &= u(x) - 92 \leq 0 \\
g_2(x) &= -u(x) \leq 0 \\
g_3(x) &= v(x) - 110 \leq 0 \\
g_4(x) &= -v(x) + 90 \leq 0 \\
g_5(x) &= w(x) - 25 \leq 0 \\
g_6(x) &= -w(x) + 20 \leq 0 \\
&\text{where} \\
u(x) &= 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 \\
v(x) &= 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 - 0.0021813x_3^2 \\
w(x) &= 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4.
\end{aligned}$$

- Box constraint: $l_i \leq x_i \leq u_i,\ i = 1, 2, \cdots, 5$, where $l = (78, 33, 27, 27, 27)$ and $u = (102, 45, 45, 45, 45)$.
- The global minimum: $x^* = (78, 33, 29.995, 45, 36.7758)$ with $f(x^*) = -30655.539$.

Problem 4.5

- Numbers of variables: 4 variables.
- Definition:

Objective function

$$f(x) = 3x_3 + 10^{-6}x_1^3 + 2x_2 + \frac{2}{3} \times 10^{-6}x_2^3$$

Inequality constraints

$$\begin{aligned}
g_1(x) &= x_3 - x_4 - 0.55 \leq 0 \\
g_2(x) &= x_4 - x_3 - 0.55 \leq 0
\end{aligned}$$

Equality constraints

$$\begin{aligned} h_1(x) &= 1000[\sin(-x_3 - 0.25) + \sin(-x_4 - 0.25)] + 894.8 - x_1 = 0 \\ h_2(x) &= 1000[\sin(x_3 - 0.25) + \sin(x_3 - x_4 - 0.25)] + 894.8 - x_2 = 0 \\ h_3(x) &= 1000[\sin(x_4 - 0.25) + \sin(x_4 - x_3 - 0.25)] + 1294.8 = 0 \end{aligned}$$

- Box constraint: $l_i \leq x_i \leq u_i,\ i = 1, 2, 3, 4$, where $l = (0, 0, -0.55, -0.55)$ and $u = (1200, 1200, 0.55, 0.55)$
- The global minimum: $x^* = (679.9453, 1026, 0.118876, -0.3962336)$ with $f(x^*) = 5126.4981$.

Problem 4.6

- Numbers of variables: 2 variables.
- Definition:

Objective function

$$f(x) = (x_1 - 10)^3 + (x_2 - 20)^3$$

Inequality constraints

$$\begin{aligned} g_1(x) &= (x_1 - 5)^2 + (x_2 - 5)^2 + 100 \leq 0 \\ g_2(x) &= (x_1 - 5)^2 + (x_2 - 5)^2 - 82.81 \leq 0 \end{aligned}$$

- Box constraint: $l \leq x_i \leq 100,\ i = 1, 2$, where $l = (13, 0)$
- The global minimum: $x^* = (14.095, 0.84296)$ with $f(x^*) = -6961.81388$.

Problem 4.7

- Numbers of variables: 10 variables.

- Definition:

Objective function

$$\begin{aligned} f(x) &= x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3-10)^2 + 4(x_4-5)^2 + (x_5-3)^2 \\ &\quad +2(x_6-1)^2 + 5x_7^2 + 7(x_8-11)^2 + 2(x_9-10)^2 + (x_{10}-7)^2 + 45 \end{aligned}$$

Inequality constraints

$$\begin{aligned} g_1(x) &= 4x_1 + 5x_2 - 3x_7 + 9x_8 - 105 \leq 0 \\ g_2(x) &= 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0 \\ g_3(x) &= -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0 \\ g_4(x) &= 3(x_1-2)^2 + 4(x_2-3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0 \\ g_5(x) &= 5x_1^2 + 8x_2 + (x_3-6)^2 - 2x_4 - 40 \leq 0 \\ g_6(x) &= 0.5(x_1-8)^2 + 2(x_2-4)^2 + 3x_5^2 - x_6 - 30 \leq 0 \end{aligned}$$

$$\begin{aligned} g_7(x) &= x_1^2 + 2(x_2-2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0 \\ g_8(x) &= -3x_1 + 6x_2 + 12(x_9-8)^2 - 7x_{10} \leq 0 \end{aligned}$$

- Box constraint: $-10 \leq x_i \leq 10, \quad i = 1, 2, \cdots, 10.$

- The global minimum:

$x^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644,$ $9.828726, 8.280092, 8.375927)$ with $f(x^*) = 24.3062091$.

Problem 4.8

- Numbers of variables: 2 variables.

- Definition:

Objective function

$$f(x) = -\frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1+x_2)}$$

Inequality constraints

$$\begin{aligned} g_1(x) &= x_1^2 - x_2 + 1 \le 0 \\ g_2(x) &= 1 - x_1 + (x_2-4)^2 \le 0 \end{aligned}$$

- Box constraint: $0 \le x_i \le 10,\ i = 1, 2.$
- The global minimum: $x^* = (1.2279713, 4.2453733)$ with $f(x^*) = -0.095825$.

Problem 4.9

- Numbers of variables: 7 variables.
- Definition:

  Objective function

$$f(x) = (x_1-10)^2+5(x_2-12)^2+x_3^4+3(x_4-11)^2+10x_5^6+7x_6^2+x_7^4-4x_6x_7-10x_6-8x_7$$

  Inequality constraints

$$\begin{aligned} g_1(x) &= 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 - 127 \le 0 \\ g_2(x) &= 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 - 282 \le 0 \\ g_3(x) &= 23x_1 + x_2^2 + 6x_6^2 - 8x_7 - 196 \le 0 \\ g_4(x) &= 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \le 0 \end{aligned}$$

- Box constraint: $-10 \le x_i \le 10,\ i = 1, 2, \cdots, 7.$

- The global minimum: $x^* = (2.330499, 1.951372, -0.4775414, 4.365726,$ $-0.6244870, 1.038131, 1.594227)$ with $f(x^*) = 680.6300573$.

Problem 4.10

- Numbers of variables: 8 variables.
- Definition:

  Objective function

$$f(x) = x_1 + x_2 + x_3$$

  Inequality constraints

$$\begin{aligned}
g_1(x) &= -1 + 0.0025(x_4 + x_6) \leq 0 \\
g_2(x) &= -1 + 0.0025(-x_4 + x_5 + x_7) \leq 0 \\
g_3(x) &= -1 + 0.01(-x_5 + x_8) \leq 0 \\
g_4(x) &= 100x_1 - x_1x_6 + 833.33252x_4 - 83333.333 \leq 0 \\
g_5(x) &= x_2x_4 - x_2x_7 - 1250x_4 + 1250x_5 \leq 0 \\
g_6(x) &= x_3x_5 - x_3x_8 - 2500x_5 + 1250000 \leq 0
\end{aligned}$$

- Box constraint: $l_i \leq x_i \leq u_i,\ i = 1, 2, \cdots, 8$, where $l = 10*(10, 100, 100, 1, 1, 1, 1, 1)$ and $u = 1000 * (10, 10, 10, 1, 1, 1, 1, 1)$.
- The global minimum: $x^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985,$ $217.9799, 286.4162, 395.5979)$ with $f(x^*) = 7049.3307$.

Problem 4.11

- Numbers of variables: 2 variables.
- Definition:

  Objective function

$$f(x) = x_1^2 + (x_2 - 1)^2$$

Equality constraints

$$h_1(x) \;=\; x_2 - x_1^2 = 0$$

- Box constraint: $-1 \leq x_i \leq 1,\ i = 1, 2.$
- The global minimum: $x^* = \pm(1/2^{0.5}, 1/2)$ with $f(x^*) = 0.75.$

Problem 4.12

- Numbers of variables: 3 variables.
- Definition:

  Objective function

  $$f(x) = 1 - 0.01[(x_1 - 5)^2 + (x_2 - 5)^2 + (x_3 - 5)^2]$$

  Inequality constraints

  $$g_{i,j,k}(x) \;=\; (x_1 - i)^2 + (x_2 - j)^2 + (x_3 - k)^2 - 0.0625 \leq 0$$

  where $i, j, k = 1, 2, \cdots, 9.$
- Box constraint: $0 \leq x_i \leq 10,\ i = 1, 2, 3.$
- The global minimum: $x^* = (5, 5, 5)$ with $f(x^*) = 1.$

Problem 4.13

- Numbers of variables: 5 variables.
- Definition:

  Objective function

  $$f(x) = e^{x_1 x_2 x_3 x_4 x_5}$$

Equality constraints

$$\begin{aligned} h_1(x) &= x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ h_2(x) &= x_2 x_3 - 5 x_4 x_5 = 0 \\ h_3(x) &= x_1^3 + x_2^3 + 1 = 0 \end{aligned}$$

- Box constraint: $l_i \leq x_i \leq u_i,\ i = 1, 2, \cdots, 5$, where $u = (2.3, 2.3, 3.2, 3.2, 3.2)$ and $l = -u$.
- The global minimum: $x^* = (-1.717143, 1.595709, 1.827247, -0.7636413, -0.763645)$ with $f(x^*) = 0.0539498$.