Global Optimization Using Piecewise Linear Approximation

by

Loay Alkhalifa

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2020 by the
Graduate Supervisory Committee:

Hans Mittelmann, Chair
Dieter Armbruster
Adolfo Escobedo
Rosemary Renaut
Jorge Sefair

ARIZONA STATE UNIVERSITY

May 2020

ABSTRACT

Global optimization (programming) has been attracting the attention of researchers for almost a century. Since linear programming (LP) and mixed integer linear programming (MILP) had been well studied in early stages, MILP methods and software tools had improved in their efficiency in the past few years. They are now fast and robust even for problems with millions of variables. Therefore, it is desirable to use MILP software to solve mixed integer nonlinear programming (MINLP) problems. For an MINLP problem to be solved by an MILP solver, its nonlinear functions must be transformed to linear ones. The most common method to do the transformation is the piecewise linear approximation (PLA). This dissertation will summarize the types of optimization and the most important tools and methods, and will discuss in depth the PLA tool. PLA will be done using nonuniform partitioning of the domain of the variables involved in the function that will be approximated. Also partial PLA models that approximate only parts of a complicated optimization problem will be introduced. Computational experiments will be done and the results will show that nonuniform partitioning and partial PLA can be beneficial.

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my advisor, Dr. Mittelmann, who guided me through my study and gave me the opportunity to do my PhD at ASU. I am very grateful to have the opportunity to explore the field of optimization under the supervision of one of the world leaders in this field. Also, special thanks to my committee who assisted writing this dissertation me through their valuable comments and suggestions.

I also would like to express my appreciation to my parents and my siblings, who kept me motivated by their prayers and supportive words. To my lovely wife and kids, Aseel and Rund, my heartfelt gratitude for being my strength in everything I have been through.

Finally, to all my friends, there are more names than what I can fit in this page, but thank you all for being around.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

Optimization (or programming) is a large field of research that takes interest in making some decisions to optimize an objective. The main components of an optimization problem are an *objective function* that needs to be optimized (minimized or maximized) by assigning values to decision *variables*, and restrictions or *constraints* that are applied on the variables. Given a vector $x = (x_1, x_2, \ldots, x_n)$, the optimization model can be generally formulated as:

$$\min\ f(x) \tag{1.1a}$$

$$\text{subject to } g(x) \geq 0 \tag{1.1b}$$

$$x \in X \subseteq \mathbb{R}^n, \tag{1.1c}$$

where $f : X \to \mathbb{R}$ and $g : X \to \mathbb{R}^m$. The optimization problem of the form 1.1 has n variables and m constraints and the goal is to minimize the function f.

The research on optimization can be divided into many areas based on different aspects. For example, based on what kind of a solution is required to solve the optimization problem, the field of study can be divided into *local optimization* and *global optimization*. The local optimization aims at finding local solutions (the smallest or largest value of the objective function over a part of the original domain). In the case of global optimization, the goal is to find a global solution, then prove it is global. The discussion in this dissertation is mostly on the global optimization.

Another important classification of optimization is based on the domains of the variables. If the research area studies problems where the variables have continuous

domains, then the area is in the *continuous optimization* field. *Discrete optimization* deals with problems that have variables with discrete domains. Integer and binary variables are special cases of discrete variables. Both classes are discussed in the following chapters.

Also there are *constrained* and *unconstrained optimization* fields that study optimization problems with or without constraints. Other classes can be classified depending on the functions in the objective or constraints, whether the functions are linear, convex, nonconvex, etc.

Although the optimization classes mentioned above can be studied separately, they are usually linked together. For example, local optimization is very helpful to do global optimization, and doing constrained optimization starts in many cases by doing it as if it is unconstrained. Therefore, even though the main discussion is on the nonconvex global optimization, all other classes are discussed thoroughly in this dissertation.

1.1 Motivation

Many research areas in engineering, science, economics, depend on optimization techniques to solve their optimization problems. Real life optimization problems arise in economics modeling, finance, networks, transportation, operation research, chemical; electrical; and civil engineering, biology, mechanics, and many other areas. Most of these problems have the most complicated form of optimization problems (nonconvex constrained global optimization). Therefore, many optimization techniques and algorithms have been introduced to solve complicated problems arising from these fields.

Although the improvement of optimization methods is notable, it is still not enough. Countless instances of problems from different optimization classes may

still be unsolved (this is either due to the size of these problems or the complexity). The number of unsolved problem is declining due to the continued improvement of software solvers, algorithms, tools, etc. Detailed discussion on this development will be presented in this dissertation.

The main contribution of the dissertation will be on one of the effective tools in nonconvex global optimization. This tool, Piecewise Linear Approximation (PLA), helps approximating a complicated problem, e.g., nonlinear, by a simpler one, linear. As PLA is used by many algorithms, improving it can improve these algorithms, and therefore a better performance of solvers can be obtained.

The existing PLA models will be modified by either modifying the models themselves or changing how to apply them to a problem. The test results show that in many cases the solvers perform better with modified models compared to the standard ones. Also with the PLA models, successful attempts were made on fully transforming nonlinear problems to linear ones, enabling linear solvers to deal with these problems.

1.2 Dissertation Outline

Each chapter provides an overview of the literature that is related to the chapter subjects. The materials of this dissertation are organized as follows: Chapter 2 presents an overview of global optimization. Different classes of problems are defined, followed by a short survey of optimization tools and techniques. After that, a summary of the most important algorithms that deal with each class is presented.

Chapter 3 concentrates on the class of mixed integer nonlinear programming (MINLP) problems. The most common methods and algorithms that handle these problems are discussed in detail. Then an overview of the global solvers that can solve this class of problems (nonconvex MINLP global solvers) is presented. The end of

the chapter provides the results of computational experiments that compare between the performances of these solvers on solving some difficult problems.

The main topic of the dissertation, PLA, is demonstrated in Chapter 4. This chapter starts by presenting the background of PLA and a summary of the most common models for both one and two dimensional cases. Then, few approaches on how to improve the models are introduced. One of the approaches shows how to take an advantage of a local solution to produce a better PLA model. Another approach is to apply the PLA on only a part of the optimization problem, which makes it easier to be handled by a solver.

Finally, the computational results are presented in Chapter 5. The PLA models were tested on two classes of nonconvex optimization: quadratically constrained programming and mixed integer quadratically constrained programming. The chapter ends by testing the models on few MINLP problems with two variables. The following chapter, Chapter 6, will provide the conclusions that summarize the main aspects of the dissertation.

Chapter 2

GLOBAL OPTIMIZATION

The goal of global optimization is to find a solution where the function value at this solution is the same as or better than the value at any feasible solution. Assume x^* is the global minimum of the function f, then $f(x^*) \leq f(x)$ for any x. The degree of difficulty of global optimization varies depending on the class of the optimization problems. In order to easily distinguish between different types of optimization problems, Model 1.1 can be reformulated to

$$\min f(x) \tag{2.1a}$$

$$\text{subject to } h(x) = 0 \tag{2.1b}$$

$$g(x) \geq 0 \tag{2.1c}$$

$$x \in X \subseteq \mathbb{R}^n \tag{2.1d}$$

$$x_i \in \mathbb{Z}, \qquad \text{for } i \in Z \subseteq \{1, 2, \ldots, n\}, \tag{2.1e}$$

where $f : X \to \mathbb{R}$, $g : X \to \mathbb{R}^m$, $h : X \to \mathbb{R}^l$ for $l \leq n$. Before listing the types of problems, two general types should be mentioned: convex and nonconvex programming. If f is convex and X is a convex set, and the constraints define a convex feasible set,then the continuous problem ($Z = \phi$) is convex. Convex problems have an important property that makes the global optimization much easier compared

to nonconvex problems. The property implies that every local solution is also a global solution, thus doing global optimization only requires finding a local solution.

2.1 Types of Optimization Problems

Assume f and all functions in the constraints in Model 2.1 are linear functions and $Z = \emptyset$, then the optimization problem is linear and it can be solved using linear optimization or programming (LP). This is the easiest optimization problem. Not only is any local solution the global one (since any linear function is convex), but also the global solution will be at one of the vertices of the polytope that is formed by the constraints. Simplex method and interior-point methods are highly efficient in finding the global optimum of LP problems.

Quadratic Programming (QP) is the second easiest type of programming after LP. The problem settings are the same, but f is a quadratic function instead of being linear. $f(x)$ is called a quadratic function if it can be written as $f(x) = \frac{1}{2}x^T Qx + x^T c$, where $Q \in \mathbb{R}^{n \times n}$ is a symmetric matrix and $c \in \mathbb{R}^n$. If Q is positive semidefinite, then f is convex, and therefore 2.1 becomes a convex QP problem which is usually as easy as a LP problem and both can be solved in polynomial time. When Q is indefinite, then the QP problem is nonconvex and the number of local solutions may increase exponentially with the number of negative eigenvalues of Q. Even though finding the global solution is hard in this case, many methods were developed to deal with this kind of problem.

When at least one of the functions in the constraints is quadratic and f is linear or quadratic, then 2.1 will be quadratically constrained programming (QCP) or quadratically constrained quadratic programming (QCQP), respectively. They are both harder than QP but they were well studied and the global optimum can be found using several methods. Convexifying the problem (bounding the nonconvex

function by a convex one) is one of the most important tools that are used by some of these methods and quadratic functions have the advantage of being easy to convexify.

The problem needs nonlinear programming (NLP) if f or at least one of the functions in the constraints is nonlinear (Z is still an empty set). NLP is considered to be harder than QCQP. One of the reasons is that convexifying an NLP problem with high degree polynomial functions is more complicated than convexifying a QCQP problem. Turning the problem into a simpler one (like a convex or LP problem) is a tool that is used by most of the algorithms that perform well on finding the global solution of NLP problems.

Now assume $Z \neq \emptyset$, Then the difficulty of the optimization problem rises to an entirely different level. Even the simplest optimization problem, LP, when some of its variables are integer, could be harder than problems that are already complex such as NLP problems. The type of optimization in this case is mixed integer linear programming (MILP). Similarly, the other types include MIQP, MIQCP, MIQCQP, and finally MINLP, which is the general case.

In the following section, we will define some concepts and introduce tools that are used by most of the global optimization algorithms that deal with the classes of optimization problems mentioned above. These algorithms play an important role in solving MINLP problems globally, which will be discussed in detail in Chapter3.

2.2 Techniques and Tools

The purpose of any technique or tool introduced in this section is to overcome one or more of the global optimization challenges. Some techniques could target the complicated functions in the problem and replace them by simple functions. Other techniques target the feasible area and divide it to many areas resulting in many

subproblems but simpler ones than the original problem. The components of most algorithms are formed by these techniques.

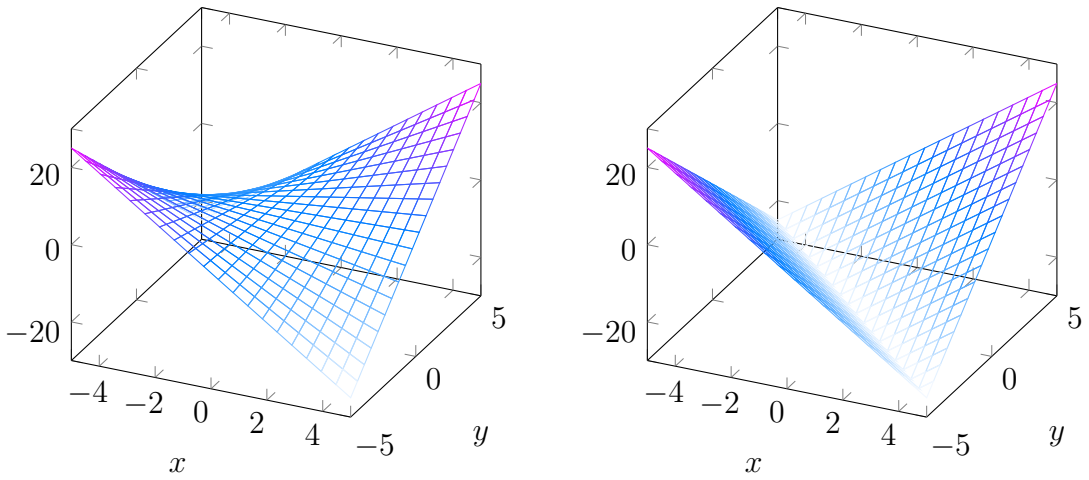
1. Divide and conquer

   Divide and conquer is a technique that recursively divides the original problem into subproblems, then finds the solution of each subproblem. The goal is to find a sequence of solutions that converges to the original problem solution. This is one of the earliest optimization techniques and it became the base of many known algorithms, such as branch and bound, and branch and select.

2. Convexification and Linearization

   This tool is the most popular tool used by global optimization algorithms. Since it is hard to deal with nonconvex problems, applying under or over-estimators to the nonconvex functions will result in a much simpler problem. If the nonconvex functions of the problem are under-estimated (in case the problem is a minimization problem) by a convex function, then the process is called convexification. Linearization is under-estimating the nonconvex function by a piecewise linear function. Assume the function $\hat{f}$ is convex (or linear), then $\hat{f}$ is an under-estimator of the function f if $\hat{f}(x) \leq f(x)$ for all $x \in X$. Once the under-estimator is found, its minimum can be found easily, and therefore can be used as a lower bound of the global minimum of the original problem.

   It is not a simple task to find a good under-estimator in general, but much research has been done in this area, and under-estimators for common nonconvex functions have been introduced. A popular example is the product function under-estimator that was introduced by McCormick (1976). Assume $f(x, y) = xy$, where $x \in [x_l, x_u]$ and $y \in [y_l, y_u]$, then f can be under-estimated



(a) $f(x, y) = xy$ (b) $\hat{f}(x, y) = \max\{5x+5y-25, -5x-5y-25\}$

Figure 2.1: An Example of an Under-estimator of a Product Function.

by the function

$$\hat{f}(x, y) = \max\{y_u x + x_u y - x_u y_u, y_l x + x_l y - x_l y_l\}.$$

Figure 2.1 illustrates this example for $(x, y) \in [-5, 5]^2$.

$f(x, y) = xy$ is a quadratic function, so it was not challenging to find a good under-estimator. However the more complex the function is, the harder the convexification gets. A detailed study about under-estimators can be found in Tawarmalani and Sahinidis (2002).

3. Separation

   In case it is hard to find a good under-estimator of a nonconvex function, separation of the function might help. A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be separable if it can be written as

   $$f(x) = \sum_{i=1}^{n} f_i(x_i),$$

   where $f_i : \mathbb{R} \to \mathbb{R}$ for $i = 1, 2, \ldots, n$. This property is useful because under-estimating is easier for a univariate function. Thus after finding a convex under-

estimator of each univariate function, the sum of these under-estimators is also a convex under-estimator of the original function.

4. Factorization

   Factorization is another useful tool to find a good convex under-estimator. The complicated multivariate function is transformed to a univariate function by introducing new variables and constraints, see McCormick (1976). For example, assume $f : \mathbb{R}^3 \to \mathbb{R}$ is defined by $f(x, y, z) = \exp(\frac{xy}{z^2})$. Then it can be rewritten as $f(w_1) = e^{w_1}$, where $w_1 = \frac{w_2}{w_3}$,$w_2 = xy$,and $w_3 = z^2$. Now the function f is univariate, and the other functions are either convex or easy to be under-estimated by convex functions.

5. Piecewise Linear Approximation

   Approximating the nonlinear functions in the objective or constraints by piecewise linear functions gives the advantage of turning an MINLP problem into an MILP problem. This approach is more useful when the number of variables is small or the function is separable. When the problem has many variables, this approach might complicate the problem. Comprehensive study on this approach is provided in Chapter 4.

6. Cuts

   Cuts (cutting planes) are additional constraints that are added to the original problem resulting in decreasing the feasible area without eliminating any feasible solution. This technique can be very powerful in simplifying MILP problems. If the problem is simple, applying enough cuts can form the convex hull, which is the smallest possible feasible area that contains all integer solutions. In this case the global solution is guaranteed to be found on the boundary of the feasible area. Although finding the convex hull of an MILP problem is next to impossible

in general, obtaining cuts is possible and they will still be extremely useful even if they do not form a convex hull.

Many methods were introduced to find good cuts and the research in this area is still improving. Examples of methods to obtain cuts are Gomory cuts, mixed-integer rounding (MIR) cuts, and cover cuts. Adding cuts to a global optimization method produces new effective algorithms that can solve more complicated problems. For example, the simplex method algorithm can only solve LP problems, but adding Gomory cuts during the steps of the algorithm allows it to solve MILP problems, see Gomory (1958).

2.3 Algorithms

Many exact algorithms arose from combining techniques, such as the ones mentioned in the previous section, with recursive strategies. In this section we will briefly describe common algorithms that are highly efficient in solving problems easier than MINLP problems. In general, they cannot do global MINLP, but they will be the main ingredients of forming MINLP algorithms.

2.3.1 Branch and Bound

The Branch and Bound (B&B) algorithm is one of the fundamental methods that are used to solve MILP problems, see Land and Doig (1960). This algorithm embodies a divide and conquer strategy. It enumerates candidate solutions that form a rooted tree and it seeks the global solution. An MILP problem with only n binary variables has up to 2^n possible solutions, which makes it impossible to evaluate them all if n is large. The advantage of this method is its ability to eliminate a huge amount of these possible solutions, as will be shown below.

To describe the Algorithm, assume the problem 2.1 is an MILP problem, i.e., f, g, and h are linear, and $Z \neq \emptyset$. The B&B algorithm starts solving this problem by ignoring the integer variables, and solves it as an LP problem. The solution will return some or all of the integer variables having fractional values (if all integer variables turned out to be integer, then the solution is the optimal solution).

After solving the LP problem, B&B chooses one of the integer variables that have fractional values (usually the variable with the smallest fractional part) to partition its domain and create two sub-problems, which we will call nodes. Assume the chosen variable is $x_j = x_j^*$, where $j \in Z$ and x_j^* is not integer. The two nodes are similar to the original node with one extra constraint to each node: one node with $x_j \leq \lfloor x_j^* \rfloor$ and the other node with $x_j \geq \lceil x_j^* \rceil$. Notice that any resulting node will have an objective value that is greater than or equal to the branched node.

Now B&B keeps branching and creating more nodes by repeating the process that was applied to the original node on every node. Whenever a node's solution returns with all integer variables having integer values, then no further branching is needed on that node and the node is *pruned* or *fathomed*. The solution now is a candidate for the global solution. The best available integer solution so far is called the *incumbent solution*. The incumbent solution is updated every time a node results in a better integer solution.

The objective value at the incumbent solution (incumbent value) is an upper bound of the optimal value. Therefore, if a node results in an objective value that is greater than the incumbent value, then branching will only result in nodes with greater objective values. Thus the node is pruned and no more branching is required on it. Another case that results in pruning a node is when the node has no feasible solution. With pruning property, the B&B method is able to eliminate big parts of

the search region in short time. When all nodes are pruned (there is no branching required on any node), then the current incumbent solution is the global optimum.

Considerable research has been done to improve the components of the B&B method. These components include search strategies (which node should be tested), branching strategies (which variable should be selected for branching), and pruning rules. Variations of these components have proved that some strategies are better than others. A recent survey by Morrison *et al.* (2016) presents more detail about these strategies.

Since it was introduced, the B&B method has been efficiently used to solve a wide range of MILP problems. Its components became the basis of many algorithms that are able to deal with all types of optimization problems, including nonconvex MINLP.

2.3.2 Outer Approximation

The outer approximation method is a good option to solve any convex optimization problem. The presence of convex functions in this kind of problems gives the advantage of easy Linearization of the functions. For any convex function $f(x)$, we have the following inequality:

$$f(x) \geq f(\bar{x}) + \nabla f(\bar{x})^T (x - \bar{x}), \tag{2.2}$$

where $\bar{x}$ is in the domain of f. Now if the optimization problem is convex, then the constraints will form a convex area which can be approximated from outside by linear functions using Inequality 2.2. Figure 2.2 illustrates this approximation in the two dimensional case.

The outer approximation algorithm was first introduced by Duran and Grossmann (1986), and it targets convex MINLP. Solving convex MINLP problems using this method starts with assigning feasible values for the integer variables. Now since the

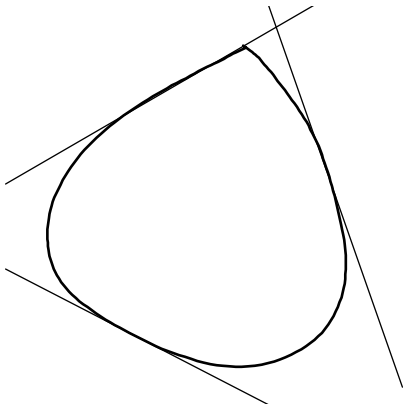


Figure 2.2: Outer Approximation of a Convex Area.

integer variables are given, the problem becomes a convex NLP, and Model 2.1 can be rewritten as

$$\min f(x) \tag{2.3a}$$

$$\text{subject to } g(x) \leq 0 \tag{2.3b}$$

$$x \in X \subseteq \mathbb{R}^n \tag{2.3c}$$

$$x_i = x_i^I, \qquad \text{for } i \in Z \subseteq \{1, 2, \ldots, n\}, \tag{2.3d}$$

where x_i^I is a fixed integer value, and f and g are convex and continuously differentiable (the equality constraints are omitted for simplicity). Now the convex NLP model is easy to solve and the resulting solution is feasible for the original problem and it will produce an upper bound of the global minimum. The NLP solution and its objective value are denoted by $\bar{x}$ and $\bar{v}$, respectively.

To obtain a lower bound of the optimal value, the original problem, the convex MINLP, can be reformulated to be an MILP problem using the NLP solution $\bar{x}$ and Inequality 2.2. By introducing an auxiliary variable α, the model is written as

$$\min \alpha \tag{2.4a}$$

$$\text{subject to } f(\bar{x}) + \nabla f(\bar{x})^T(x - \bar{x}) \leq \alpha \tag{2.4b}$$

$$g(\bar{x}) + \nabla g(\bar{x})^T(x - \bar{x}) \leq 0 \tag{2.4c}$$

$$x \in X \subseteq \mathbb{R}^n \tag{2.4d}$$

$$x_i \in \mathbb{Z}, \quad \text{for } i \in Z \subseteq \{1, 2, \ldots, n\}. \tag{2.4e}$$

The outer approximation methods depend on the property that all optimal solutions of the original model are optimal for Model 2.4 with the same objective values, see Duran and Grossmann (1986); Fletcher and Leyffer (1994); Bonami *et al.* (2008).

Solving this MILP model will produce a lower bound of the optimal value, which is denoted by $\underline{v}$. The global optimal value must be between $\bar{v}$ and $\underline{v}$. Now the values of the integer variables are updated from the MILP solution, and they can be used as new x_i^I in 2.3d. Then 2.3 is solved again to get a new $\bar{x}$ and update $\bar{v}$ if it is better. The new $\bar{x}$ will lead to more linearizations so solving 2.4 might lead to an improvement in $\underline{v}$ value.

The algorithm keeps alternating between solving the Models 2.3 and 2.4 and the values $\bar{v}$ and $\underline{v}$ are updated. When the difference between $\bar{v}$ and $\underline{v}$ is arbitrarily small, then $\bar{v}$ is the global optimal value and $\bar{x}$ is the global optimum.

Similar to the B&B method, the outer approximation method cannot deal with nonconvex MINLP problems. However, this method is an important component of most of the MINLP algorithms, as we will see in Chapter3

2.3.3 Cutting Plane Algorithms

Many algorithms have benefited from cutting planes and became able to solve optimization problems that cannot be solved using these algorithms without cuts.

One of the earliest algorithms is Gomory's algorithm, see Gomory (1958), which can use the simplex method to solve MILP problems. The procedures of solving an MILP problem starts by using the simplex method to solve the problem as an LP problem. Then a Gomory cut is applied by using a constraint that violates the integrality of one of the integer variables in the optimal tableau.

To illustrate the step of adding the Gomory cut, consider the following numerical example. Assume the constraint

$$1.3x_1 + 2.8x_2 + 0.5x_3 + x_4 = 3.7$$

is in the optimal tableau, and all variables are non negative integer. If the fractional part of the coefficients is split from the integer part then the left hand side will be $x_1 + 2x_2 + x_4 + 0.3x_1 + 0.8x_2 + 0.5x_3$. Now the sum of the first three terms has to be integer, so the fractional part of the right hand side results from the sum of the last three terms. Therefore, the Gomory cut in this case is the inequality

$$0.3x_1 + 0.8x_2 + 0.5x_3 \geq 0.7.$$

In general, if a constraint is given by $\sum_i^n a_i x_i = b$, then the Gomory cut is

$$\sum_i^n f_i x_i \geq f,$$

where $f_i = a_i - \lfloor a_i \rfloor$ and $f = b - \lfloor b \rfloor$.

After adding this inequality to the constraints of the original problem, the solution obtained by the simplex method is no longer feasible. Therefore, the simplex method is applied again to the LP problem with the extra constraint. A new solution is obtained and a new cut is added. These steps are repeated until the simplex method produces a solution that does not violate the integrality constraints, and it is the global solution. If all variables are integer, this method is guaranteed to produce the optimal solution in a finite number of steps.

Cutting plane methods are not limited to MILP problems. For example, the extended cutting plane method by Westerlund and Pettersson (1995) can deal with convex MINLP, which the Gomory method cannot deal with. This method is an extension to the Kelley cutting plane method that was introduced by Kelley (1960) to solve convex NLP problems. Its algorithm is similar to the outer approximation algorithm that was explained in the previous section. The difference here is that the extended cutting plane method does not solve the NLP relaxation during the iterations. It keeps adding linear cuts and solving the MILP relaxation until a solution that satisfies the nonlinear constraints within an ϵ tolerance is found. The objective value at this solution is at most ϵ below the objective value at the global optimal solution.

The cuts that are added in every iteration of the extended cutting plane and Kelley's methods are different from Gomory cuts. The cuts are obtained by constraints similar to Inequality 2.2 for convex functions.

Another example of cuts is cover cuts, which can be used to significantly improve the performance of algorithms for most kinds of optimization problems. These cuts are easily obtained from the constraints of the original problem. For example, assume x_1, x_2, and x_3 are binary variables and $3x_1 + 4x_2 + 8x_3 \leq 10$ is one of the constraints. Few possible cuts can be obtained from the constraint such as $x_1 + x_3 \leq 1$ and $x_2 + x_3 \leq 1$. For more examples of cuts and cutting planes algorithms, Belotti *et al.* (2013) presented a detailed study about the subject.

Chapter 3

MINLP

The most difficult class of optimization is MINLP, where the model takes the form of Model 2.1 with f,g, and h being at least twice differentiable and nonconvex. Having both continuous/discrete variables and nonlinear functions allows most of deterministic real life problems to be modeled and solved using MINLP. However, even the simplest example of MINLP problems is considered to be an NP-hard problem. Therefore, with the increasing need for MINLP in many applications, this area has been a target for researchers for few decades. The development of MINLP algorithms and computational methods increased dramatically in the past few years, and handling global MINLP problems in real life applications is promising.

3.1 Background

The promising results of methods and algorithms that were introduced to deal with the easier classes of optimization problems encouraged researchers to extend these methods and apply them to solve MINLP problems. Early attempts take advantage of methods such as B&B, outer approximation, and cutting planes and use them on MINLP problems with a small number of variables. Horst (1990) presents a review of these methods. Unfortunately, if a solution is found by any of these methods, it cannot be proven to be global, also the methods are not effective in the presence of integer variables and nonconvex functions. Therefore, further improvement on MINLP algorithms is needed.

A major improvement in the MINLP research area was when Falk and Soland (1969) applied the B&B method on separable NLP problems. Later, McCormick

(1976) introduced a B&B algorithm that deals with general nonconvex NLP problems. This algorithm has the same divide and conquer principles of B&B described 2.3.1. Branching is applied to a continuous variable by partitioning its domain and creating a subproblem for each part, and the lower bound of the function's optimal value is obtained by solving a convex or linear relaxation of the original problem. This algorithm later came to be known as the spatial branch and bound (sB&B) algorithm and became one of the most significant algorithms that deal with MINLP problems. More details about sB&B are presented in the following section.

The outer approximation algorithm is also modified to deal with nonconvex MINLP problems. One of the most successful attempts to do this modification was presented by Viswanathan and Grossmann (1990). Kesavan *et al.* (2004) presented a more recent outer approximation method that deals with nonconvex MINLP problems whose functions can be separated into integer variable functions and continuous variable functions.

3.2 Spatial Branch and Bound (sB&B)

After the sB&B algorithm was first introduced to solve NLP problems, it was later realized that it can be applied to MINLP problems, as mentioned for example by Smith and Pantelides (1997). It is possible to use the B&B algorithm to solve MINLP problems and the global optimum will be found eventually, but performing nonconvex NLP at every node would be very expensive. On the other hand, sB&B requires either convex NLP or LP at every iteration, as will be explained below.

To describe the algorithm, the following notations are needed: Let R_j be the j^{th} search area (region) to be tested. Let L_j and U_j be the lower and upper bounds of the optimal value of the objective function on R_j. The algorithmic steps are described as follows:

- **Step 1: Initialization:**

  Define $U := \min_j U_j$ to be the incumbent value and set $U = \infty$. Define a convergence tolerance $\epsilon > 0$. Set the whole domain of the original problem as R_1 to start the search on it and divide it later to get more subproblems.

- **Step 2: Region Selection:**

  Select a region R_j that has not been tested yet. The region selection can be done in many ways. One popular way is to select one of the regions with the smallest lower bound computed in the previous iteration. After selecting a region, go to Step 3.

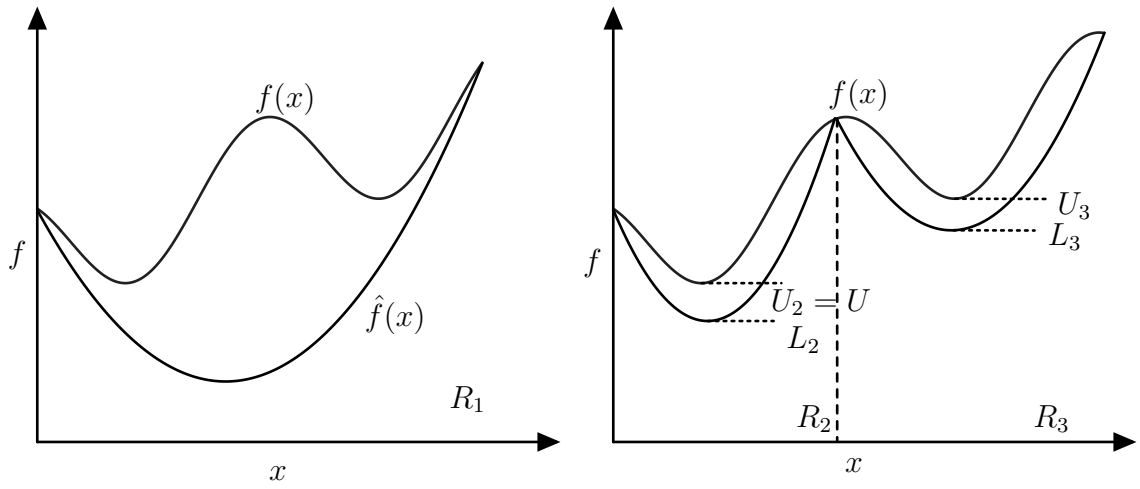
- **Step 3: Lower Bounding:**

  Apply some relaxations to the original problem in the selected region R_j. This can be done by applying linear or convex under-estimators to the nonconvex terms in the original problem. Factorization and separation are applied by most of sB&B algorithms to facilitate the process of finding a good under-estimator. The optimal value of the relaxed problem in R_j is L_j, and it will be a lower bound of the optimal solution in R_j. In Figure 3.1a, the nonconvex objective function, f, is under-estimated by the convex function $\hat{f}$.

  If the relaxed problem is linear then it will be easy to handle, but if it is non-linear (but convex) then convex programming is needed. Outer approximation algorithms are usually used by sB&B algorithms during this step.

- **Step 4: Upper Bounding:**

  Many ways can be used to find the upper bound U_j. It can be found using any local solver to find a local solution to the problem in R_j. Another way is to use the solution of the relaxed problem if it is feasible. U is updated as needed.



(a) Before Branching. (b) After Branching.

Figure 3.1: Finding Lower and Upper Bounds of the Objective Function Values in the Search Regions.

- **Step 5: Check Solution:**

  If $U_j - L_j \leq \epsilon$, then U_j is the global optimal value at R_j. The region R_j is pruned. If $U = U_j$, then the solution $x^* = x_j^*$ associated with U_j is the incumbent solution. Also the region R_j is pruned if it has no feasible solution, or if the lower bound of the optimal value is larger then the incumbent value, i.e., $L_j > U$. In Figure 3.1b, $L_3 > U_2 = U$, so the global optimal solution cannot be in the region R_3, therefore the region is pruned.

  After checking the solution, if the region is pruned and there exist at least one region that has not been tested, then go to Step 2. If there are no more regions to be tested, then the current incumbent solution is the global optimum.

  If $U_j - L_j > \epsilon$, then the optimal solution of the region R_j is not found; hence further branching is needed to tighten the bounds. Go to Step 6.

- **Step 6: Branching:**

  The region R_j is divided into two or more regions by partitioning the domain of one of the variables in the vector x. The branching variable and the branching point are not chosen randomly. Many branching strategies can be used to determine how to branch R_j. For example, if the solution x_j^* has a noninteger value for an integer variable, then it is always a good idea to branch on this variable. After branching the region, go to Step 2.

Finding good upper and lower bounds may spare much time during the solution process. For example, in the lower bounding step, the integrality constraints can be ignored, resulting in a convex NLP or LP problem, which will make the relaxed problem easy to solve, but the resulting lower bound may not be tight, which will lead to more search regions being tested. On the other hand, if the integrality constraints are not ignored, the lower bound will be better than the one produced by the other case, and hence, many fewer search regions are explored. This advantage will come at the price of solving a convex MINLP or MILP problem, which is more expensive than convex NLP problems. The same logic applies to the upper bounding. A good upper bound can be found using an expensive method, but it will help reducing the search time.

Branching strategies were also investigated to aid finding better bounds or reducing the search area. One of the strategies is the *strong branching* strategy, which was introduced by Applegate and Bixby (1995) and Applegate *et al.* (1998) to deal with the travelling salesman problem (TSP) problem (MILP). Using this strategy, for every branching variable candidate, a branching point is selected, then the two resulting problems are solved. Some variables may result in pruning one or both nodes. Others may result in improving the lower or upper bound of the objective value. Based on these results, the branching variable is selected. Although this strat-

egy may considerably decrease the search area, the solving time may increase because of the computations that are required to solve two subproblems for many variables before branching. An alternative strategy that avoids these computation is the *pseudocosts branching* strategy (Bénichou *et al.* (1971)). The improvements of the bounds for any branching variable are estimated by keeping track of all improvements gained by earlier branchings on that variable. *Reliability branching*, introduced by Achterberg *et al.* (2005), is another strategy that combines the previous two strategies to avoid their drawbacks and exploit their benefits. A discussion on bound tightening and branching strategies for the MINLP case was given by Belotti *et al.* (2009).

Although this method guarantees global optimality, it will not produce the solution quickly most of the time, so further advances are needed. Improvements of the strategies of the steps are made, and additional steps are added. Consequently, many sB&B-based algorithms are developed, and in the following sections, the details about some of these algorithms are discussed.

3.2.1 Branch and Reduce

The branch and reduce (B&R) algorithm is a sB&B-based algorithm that was introduced by Ryoo and Sahinidis (1995) and further discussed in Ryoo and Sahinidis (1996). This algorithm resulted in a significant improvement in the sB&B method. In addition to the steps performed in the sB&B method, this algorithm adds variable domain reduction (bound tightening) steps before the lower bounding step and after the upper bounding step. If the domain reduction is successful for at least one variable, then steps 3 and 4 are repeated, since the optimal value bounds are likely to be improved.

The goal of the domain reduction is to change the domain of a variable x_j from $[l_j, u_j]$ to $[l'_j, u'_j]$, where $l_j < l'_j$ and $u'_j < u_j$ (changing either the upper or the lower

bound is enough). If new variables are introduced due to factorization, then their domains are also subject to reduction and this reduction can be very useful.

One of the benefits of domain reduction is related to the under-estimators of the nonconvex functions in the problem, especially the linear under-estimators. The domain reduction of an auxiliary variable will lead to a tighter linear under-estimator. The tighter the under-estimator is, the better the lower bound of the optimal value, and hence, less time is needed until the global minimum is found.

The benefit of domain reduction can go beyond improving the bounds of the optimal value in a region. In some cases, reduction will result in fathoming a search region, instead of just improving its bounds. This happens when the reduction of the domain of a variable in the search region leads to an infeasible subproblem. Also if the objective function is factorable and replaced by the auxiliary variable x_k whose domain is reduced to $[l'_k, u'_k]$, for $k > n$, l'_j might be larger than U. As a result, the search region is fathomed, thus sparing the time that would be used on solving the relaxed problem.

The impact of domain reduction on the sB&B method resulted in including domain reduction steps in most of the sB&B algorithms. There are many techniques that can be applied to reduce the domain of a variable. Two common techniques are optimality-based (OB) and feasibility-based (FB) tests, that were discussed by Ryoo and Sahinidis (1995). More recent details about OB and FB and other domain reduction techniques can be found in Belotti *et al.* (2013), who use the term *bound tightening* to refer to domain reduction.

3.2.2 α Branch and Bound

Another algorithm that is based on sB&B is αBranch and Bound (αB&B), which was introduced by Androulakis *et al.* (1995) for NLP problems, inspired by the algo-

rithm proposed by Maranas and Floudas (1994). The algorithm was extended later by Adjiman *et al.* (2000) to deal with MINLP problems. With domain reduction steps, the steps of this algorithm are similar to those of the B&R algorithm, but the way these steps are done might be different.

One of the biggest advantages of the αB&B algorithm is its capability of finding a lower bound of the optimal value (Step 3) for nonfactorable problems. Creating good under-estimators does not require factorization in the αB&B algorithm. Although factorization helps finding linear or convex under-estimators, it requires introducing new auxiliary variables, which can be avoided here.

αB&B constructs the under-estimators of special cases of nonconvex functions (bilinear, trilinear, fractional,... etc.) using the convex envelopes that already existed, and they are summarized in Adjiman *et al.* (1998). For a general nonconvex function $f(x)$, where $l \leq x \leq u$ for $l, u \in \mathbb{R}^n$, the convex under-estimator constructed by this algorithm is given by

$$f(x) - \sum_{i=1}^{n} \alpha_i (x_i - l_i)(u_i - x_i), \tag{3.1}$$

where α_i, for $i = 1, 2, \ldots, n$, are real positive scalars that make the matrix $H_f(x) + 2\text{diag}(\alpha_i)$ positive semidefinite, where $H_f(x)$ is the Hessian matrix of $f(x)$. Choosing α_i's that produce a good under-estimator is not simple. If the α_i's are large enough, the matrix $H_f(x) + 2\text{diag}(\alpha_i)$ will be positive semidefinite, and therefore, the function (3.1) is convex. However, it is not guaranteed that (3.1) will be a good under-estimator, especially if the α_i's are very large. A detailed study of strategies on how to choose α_i is given by Adjiman *et al.* (1998).

Since the Hessian matrix of a nonconvex function is required to construct its convex under-estimator, the nonconvex functions of an optimization problem must be twice differentiable, if the problem is solved using the αB&B algorithm. Also using the function (3.1) to under-estimate the nonconvex part, restricts the αB&B

algorithm to solving a convex NLP problem in Step 3, instead of having the LP option as a possible relaxation.

3.2.3 Other sB&B-based Algorithms

The promising results of sB&B-based algorithms encouraged the introduction of more algorithms that deal with MINLP problems. The core components of these algorithms are lower and upper bounding, region fathoming, and branching. The strategies on how to do each step and the addition of some optional steps, make these algorithms different from each other. For example, as mentioned earlier, adding domain reduction steps resulted in B&R algorithms, and αB&B is a result of a lower bounding strategy. In this section, a few sB&B-based algorithms are briefly discussed.

Branch and Cut (B&C) algorithms were introduced initially to solve MILP problems. Cutting planes methods are applied during B&B iterations, resulting in the B&C algorithms, more detail can be found in Padberg and Rinaldi (1991). A version of the B&C algorithm that deals with MINLP problems is introduced by Kesavan and Barton (2000) after combining cutting planes methods with sB&B methods. After the lower and upper bounding steps, cuts are generated and the new constraints are added to all search regions. Different types of cuts are generated depending on the relaxed problem, the original problem, the current solution, etc. For example, if the problem is transformed to an MILP problem, Gomory cuts or MIR cuts might be added.

Adding cuts at each iteration of the algorithm might lead to fewer search regions to be explored. However, a major drawback of this method is that the size of the relaxed problem increases exponentially since new constraints are added at every iteration. Therefore, developing strategies that assist with adding and removing cuts as needed during the algorithm is required.

Another sB&B variation is the algorithm proposed by Smith and Pantelides (1997, 1999). A symbolic reformulation step is applied before starting a similar B&R algorithm. This step, which is similar to factorization, transforms the original problem to an equivalent standard problem with linear functions or nonlinear ones whose convex under-estimators are found. It was shown that the reformulation is not affected by the presence of integer variables.

To reformulate a problem of the form (2.1) to the standard form, it is necessary to introduce k auxiliary variables, and to replace the objective function by the last auxiliary variable: $f(x) = x_{n+k}$. The resulting model is completely equivalent to the original model, and is given by

$$\min\ x_{n+k} \tag{3.2a}$$

$$\text{subject to } x_i = \Phi_i(x), \qquad \text{for } i = n+1, n+2, \ldots, n+k \tag{3.2b}$$

$$l_i \leq x_i \leq u_i, \qquad \text{for } i = 1, 2, \ldots, n+k \tag{3.2c}$$

$$x_i \in \mathbb{Z}, \qquad \text{for } i \in Z \subseteq \{1, 2, \ldots, n\}, \tag{3.2d}$$

where Φ is a function of two variables (addition, product, etc. functions) or one variable (log, trigonometric, etc, functions); and the vector $x \in \mathbb{R}^{n+k}$ includes the original variables $x_1, \ldots, x_n$ in addition to the auxiliary variables $x_{n+1}, \ldots, x_{n+k}$. The domains of the variables in Equation 3.2c are given from the set X for the first n variables, and are constructed through the reformulation process for the last k variables. The constraints $g(x) \geq 0$ and $h(x) = 0$ are included in the definitions of the auxiliary variables or in the variable domains. For illustration, assume the

following problem is to be reformulated:

$$\begin{aligned} &\min\ x_1 + x_1 x_2 \\ &\text{subject to}\ \cos x_1 + x_2^2 \leq 4 \\ &0 \leq x_1 \leq 10 \\ &-5 \leq x_2 \leq 5. \end{aligned}$$

The equivalent reformulated problem would be

$$\begin{aligned} &\min\ x_7 \\ &\text{subject to}\ x_3 = \cos x_1 && -5 \leq x_2 \leq 5 \\ &\qquad x_4 = x_2^2 && -1 \leq x_3 \leq 1 \\ &\qquad x_5 = x_3 + x_4 - 4 && 0 \leq x_4 \leq 25 \\ &\qquad x_6 = x_1 x_2 && -5 \leq x_5 \leq 0 \\ &\qquad x_7 = x_1 + x_6 && -50 \leq x_6 \leq 50 \\ &\qquad 0 \leq x_1 \leq 10 && -50 \leq x_7 \leq 60. \end{aligned}$$

Note that the constraint in this example before reformulation is replaced by the definition of the variable x_5 and its domain. It can be observed that the reformulation will immediately produce a lower and upper bound of the objective function value, which is x_7 in this example. Thus, applying domain reduction on this variable might help fathoming many search regions during the sB&R algorithm.

One of the disadvantages of this algorithm is that introducing many new variables will increase the size of the problem. However, the benefits of reformulation overcome the disadvantages. It was shown in Smith and Pantelides (1999) that this is not a major issue. In fact, the performance of sB&R algorithms might be better with reformulation.

3.3 Global MINLP Solvers

The development of optimization solver software is analogous to the development of the techniques and algorithms that solve optimization problems. It was initiated by the development of solvers for simple optimization problems like LP problems. Then, using combinations of the methods and techniques described above, many solvers have been introduced for any class of optimization problems. One of the earliest software packages that solves the MINLP problems is Sciconic, developed in the 1970s. It does not solve the MINLP problem globally, but it solves a linear approximation by turning the nonlinear low dimensional functions into piecewise linear functions using a special ordered set (SOS) of the types 1 and 2 (this technique will be discussed in detail in Chapter 4). More detail about the software can be found in Forrest and Tomlin (2007).

After that, more MILP and convex MINLP solvers were used to solve nonconvex MINLP. These solvers are based on the methods described above. Solvers based on B&B methods include Bonmin (in B-BB mode), CPLEX, fminconset, Knitro, LindoBB, MILANO, MINLP BB, MOSEK, SBB, and XPRESS. AOA, Bonmin (in B-OA mode), and DICOPT are examples of solvers that are based on outer approximation methods, whereas AlphaECP is a solver that uses the extended cutting planes method. BONMIN (in B-QG mode) and FilMINT also deal with only convex problems and they are based on linearization in the B&C algorithm. Most of theses solvers can find good solutions to MINLP problems but the global optimality is, in general, not guaranteed. Also some solvers will not attempt to solve the problem if it is not convex.

With the introduction of MINLP algorithms in the 1990s, many solvers that guarantee global optimality were developed to solve nonconvex MINLP problems. These

solvers combine the techniques outlined in the previous sections, and the majority of them are sB&B-based algorithms with domain reduction and lower bounding techniques. Also, factorization is commonly applied. Reliable MILP or convex NLP solvers are used by the global solvers to solve linear or convex relaxations through the iterations. Examples of nonconvex MINLP solvers are alphaBB, BARON, Couenne, LaGO, LINDO, ANTIGONE, and SCIP. More details about five global MINLP solvers are discussed in the following sections. Recent discussions about MINLP solvers and their development can be found in Trespalacios and Grossmann (2014) and Bussieck and Vigerske (2010).

3.3.1 BARON

BARON (Branch And Reduce Optimization Navigator) is the earliest global MINLP solver and one of the most powerful ones. It was introduced by Sahinidis (1996) and it employs the sB&R algorithm introduced around a year earlier. Later, it was further developed by Tawarmalani and Sahinidis (2005). BARON starts solving an MINLP problem by factorizing the problems using the techniques explained in Ryoo and Sahinidis (1995, 1996) or McCormick (1972, 1976). This is followed by primal heuristics and presolve methods to find a feasible solution and apply some domain reduction if possible. After that, BARON finds a valid lower bound of the optimal objective values by solving a convex relaxation of the problem obtained by under- and over estimating nonconvex terms in the problem.

The main loop of BARON is the same loop described in the sB&R algorithm. After a node is selected, FB tests are applied followed by finding a lower bound, then OB tests are applied. For each node, BARON has several options on how to find a lower bound. It can be based on an LP, MILP, or NLP relaxation. BARON has a dynamic strategy that allows it to choose whether it uses a relaxation that can be

solved easily but produces a weak lower bound (LP), or an expensive relaxation that results in a tight lower bound (MILP).

Studying the lower bounds obtained by an MILP relaxation of the parent nodes will help estimating the MILP lower bound of the current node. This can be very useful to compare the estimated value with the incumbent value, so if there is a potential in pruning the current node, then it is worth while to use an MILP relaxation, otherwise an LP relaxation is enough. Insights into the BARON strategy for choosing the lower bounding method is given in the recent study by Kılınç and Sahinidis (2017).

The solving time is influenced by the branch-and-bound tree size. The tree size is affected by branching methods and node selection strategies. The priority in choosing a branching variable is for integer variables. The branching points are the floor and ceiling value of the chosen integer variable. If the selected branching variable is continuous, then the branching point can be a convex combination between the value and the variable domain midpoint. BARON has a strategy to switch between node selection rules. The selected node could be the node with the best lower bound, the node with minimum infeasibility, based on depth first, or other selection rules. Details about branching and node selection methods in BARON can be found in Tawarmalani and Sahinidis (2002, 2004).

BARON can handle nonlinear binary functions and most unary functions, including $\log(x)$, $\exp(x)$, x^α, and α^x, for $\alpha \in \mathbb{R}$. The trigonometric functions are not supported by BARON. This disadvantage did not prevent BARON from being one of the best global MINLP solvers. In fact, comparisons between MINLP solvers show that, in different kinds of MINLP problems, BARON mostly performs better than other solvers in term of speed and number of solved test problems (even though it does not handle trigonometric functions).

3.3.2 Couenne

Couenne (Convex Over- and Under-ENvelopes for Non-linear Estimation), introduced by Belotti *et al.* (2009), is another MINLP solver that guarantees global optimality. Couenne is also based on sB&B algorithm and implements reformulation, relaxations, domain reduction, and upper bounding techniques to solve MINLP problems. Couenne starts solving the problem by reformulating it to take a simpler form to the Model 3.2.

The main loop in Couenne starts by applying a domain reduction step. This is done by several techniques including OB and FB tests. FB tests are performed at every node, since they are easy to do and might result in useful bound reductions. However, the reduction is usually weak, so other strong tests are performed . Couenne also uses a technique that is known as Aggressive FB. This technique requires applying FB tests several times for each variable in the node and might require doing NLP, see Belotti *et al.* (2009) for more detail. They produce better bounds than FB tests, but, unlike FB tests, OB and aggressive FB are expensive, so they are not called at every node. Couenne controls when to use each method. Another implemented method is reduced-cost domain reduction, which was explained in Ryoo and Sahinidis (1996). If the bounds stop improving and the node is still feasible, then Couenne moves to the lower bounding step.

The relaxation that is used by Couenne to find a lower bound of the optimal objective value in the current node is a linear relaxation. After all nonlinear terms in the problem are under-estimated by linear functions, a lower bound can be found by solving the LP problem. Before moving to the branching step, Couenne improves the lower bound by creating new linear relaxations resulting from adding more linearization inequalities. Different approaches to add linear inequalities are discussed

in Belotti *et al.* (2009). Couenne repeats this step until the solution of the linear relaxation is feasible for the original problem or the specified maximum number of iterations is reached.

When the lower bound refining is done and the node is still feasible and its global optimum is not found, then the node needs branching. For choosing the branching variable in Couenne, the selection priority is for violated integer variables. If all integer variables have integer values, then Couenne selects a continuous variable using different strategies such as strong branching, an extended version of reliability branching (Belotti *et al.* (2009)), and Violated Transfer (Tawarmalani and Sahinidis (2004)).

After selecting a branching variable, the branching point is selected based on several strategies. The most trivial strategy is to branch on the selected variable value. However, this strategy could result in an unbalanced tree, especially if the value is close to the variable bounds, so branching will produce two nodes: one of them will be similar to the current node. A better strategy is to use a convex combination between the value and domain midpoint (similar to the one used in BARON) to produce the branching point. Alternatively, Couenne chooses the branching point based on the nonlinear function of the selected variable. The point is selected such that in both new nodes, the distance between the function and its linear over- or under-estimator is the same. Both strategies produce more balanced trees than the first one.

Even though Couenne may perform worse than other global MINLP solvers in terms of number of problems that can be handled, it has an advantage over BARON in dealing with trigonometric functions. Also tests by Belotti *et al.* (2009) show that Couenne can perform better than BARON in some instances.

3.3.3 SCIP

One of the recently introduced global MINLP solvers is SCIP (Solving Constraint Integer Programs). This solver was initially introduced to solve MILP by Achterberg (2004, 2009), then it was extended by Berthold *et al.* (2012) to solve MINLP problems. It implements a sB&B algorithm with domain reduction steps and uses reformulation and linear relaxations for the lower bounding step. SCIP has numerous unique tools and the most important tool is its *constraint handler*. Each constraint handler in the solver deals with a certain class of constraints, such as linear constraints, quadratic constraints, etc. The task of a constraint handler is to check if the solution of the LP relaxation is feasible for all of its constraints. If some of the constraints are not satisfied, the handler decides whether the next step is to do domain reduction, add another constraint, or branch. Also, the constraint handler has other features that help decreasing the computation time. Vigerske (2013) presents details about the constraint handler and other tools implemented by SCIP.

Similarly to BARON, before selecting a node, SCIP performs primal heuristics and presolve techniques to find feasible solutions, reduce the search area, and obtain initial bounds for the optimal value. After that SCIP starts the main loop by selecting a node and then performing domain reduction. In addition to FB and OB tests, SCIP also uses other methods such as forward and backward propagation, where they are applied taking advantage of the problem reformulation, see Vigerske (2013). Also to avoid the complexity of applying OB tests, SCIP uses an approximation obtained by a valid inequality called *Lagrangian variable bound*, Gleixner and Weltge (2013). This approximation gives bounds that are close to the bounds obtained by OB tests, and it is much faster than an OB test.

The lower bound is obtained by solving a linear relaxation, generated by an outer approximation method. The LP problem is solved by the built in solver SoPlex. Before moving to the branching step, SCIP improves the lower bound by performing pricing and cut loops. The pricing loop regulates adding the auxiliary variables to the node by calculating the reduced costs (the amount by which the objective value will improve) of the variables and adds the ones with negative reduced cost to the node. The LP relaxation is then solved again. The cut loop, on the other hand, aims at adding valid inequalities to tighten the search area or prove infeasibility. The loops keep running until the lower bound stops improving (or the improvement is not significant) or the node becomes infeasible.

In the branching process, the candidate branching variables are all the variables from the LP solution that violate constraints, whether they are integer or continuous variables. SCIP mostly applies reliability branching and pseudocost branching, but it also applies other techniques that are used by other solvers. The branching point is mostly the variable value from the LP solution unless the value is close to one of the bounds; in this case, the branching point is shifted towards the middle of the variable domain.

The numerous techniques and plugins implemented in SCIP make it suitable for handling all kinds of optimization problems. Also most of nonconvex functions can be handled by SCIP. Like BARON, trigonometric functions are not supported. Recent discussion on SCIP and its development is given by Vigerske and Gleixner (2018).

3.3.4 LINDO

Another global MINLP solver is LINDO, and it was developed by Lin and Schrage (2009). LINDO implements a B&C algorithm, where the lower bound is obtained by solving a linear relaxation. MILP and NLP relaxations are also solved in LINDO

relying on other solvers like MOSEK and CONOPT. In the beginning of the solution process, LINDO finds a good feasible solution by a multi-start built in feature. This feature performs local optimization several times using different starting points, which will lead to many good local solutions and LINDO will pick the best one. If the problem is not complicated, this solution is often the global solution, But LINDO does not guarantee the globality at this point. To confirm the globality, it has to go through similar steps performed by other global solvers.

After selecting a node, a reformulation of the original problem is applied by LINDO to create a linear relaxation. If linearization is not possible, or if it does not give a good approximation to the nonlinear part, convex relaxations are used. One of the useful features in LINDO is its ability to recognize whether the function is convex, concave, quadratic, etc. This helps the solver making a quick decision about the type of relaxation that should be used. After solving the relaxation, cut loops are are used by LINDO to improve the lower bound.

After solving the relaxation, LINDO branches on one of the variables that violate some of the constraints. In particular, the furthest variable from its bounds is usually the branching variable. Also variables with small domains are not considered by LINDO for branching. The branching point is selected depending on the functions in the problem. For some functions, dividing the domain of the variable at certain points will create functions that are much easier to handle. For example, the absolute value function $f(x) = |x|$ is nonsmooth, but splitting the domain of x to $x \leq 0$ and $0 \leq x$ will create two smooth linear functions. Another example is trigonometric functions; they can be transformed to convex or concave functions by dividing the domain of the variable at the points $i\pi$, for $i \in \mathbb{Z}$.

LINDO can handle most types of functions, including trigonometric functions. It also supports logical statements, such as *if, and, or*, etc. One of the drawbacks of this

solver is that it does not have variable domain reduction steps as a main component of its algorithm. Adding these steps might lead to potential improvement of the solver performance.

3.3.5 ANTIGONE

ANTIGONE (Algorithms for coNTinuous/Integer Global Optimization of Nonlinear Equations) was developed by Misener and Floudas (2014) after upgrading the MIQP solver Glomiqo by Misener and Floudas (2013). The algorithm used to solve the MINLP problems is a B&C algorithm. A reformulation step is performed before entering the main loop. ANTIGONE does not do a symbolic reformulation (Smith and Pantelides (1999)) as introducing auxiliary variables is not preferred. ANTIGONE reformulates the original problem to an equivalent problem that only contains terms of special structure recognized by the solver. For each term, ANTIGONE learns the variables involved, convexity and concavity of the term along the parts of the interval, linearization or convexification of the term, derivatives, etc. This is an important step that helps the acceleration of the computation process before or during the main loop. Finally, if it is complicated to transform some terms to ones recognized by ANTIGONE, it introduces auxiliary variables to overcome the complexity of the term.

The main loop starts by solving a relaxation on the selected node, then it performs domain reduction steps before branching or fathoming the node. Many strategies are used to produce the relaxed problem depending on the structure of the term, such as Reformulation-Linearization Technique (RLT) (Sherali and Alameddine (1992)), convex envelopes by McCormick (1976); Al-Khayyal and Falk (1983), and the αB&B under-estimators. Outer approximation methods are also used to linearize some of the convex terms. If ANTIGONE detects that the problem is convex at the selected

node, it will call a local solver to solve the problem and the feasibility of the solution is checked. If the solution is not feasible, or if the convexity is not detected, then the LP relaxation is solved. After solving the LP problem, cuts are added and the problem is solved again, until the LP solution stops improving. after that, integrality is enforced and the MILP is solved. ANTIGONE uses CPLEX to solve both LP and MILP relaxations and obtain, the lower bound of the node's optimal value.

After solving MILP, domain reduction is applied. Antigone uses methods that are used by other solvers, e.g., FB and OB tests, and reduced cost domain reduction. FB tests are performed at every node since they are cheap. On the other hand, OB tests are applied on the root node. Then Antigone continues to apply them on the children nodes until the tests fail to reduce the domains of the variables. In this case, Antigone stops applying the OB tests on the children nodes of the node that failed to improve. Reduced cost domain reduction is applied on the root node after each solve of the MILP problem and the improved variables bounds (if any) are stored for the rest of the nodes.

When the node needs branching, Antigone does not branch on the integer variables because the integrality constraints are already satisfied since Antigone solves an MILP relaxation. To select a branching variable, Antigone usually uses reliability branching. The branching point is selected based on the same convex combination formula that is used by BARON and Couenne, with a condition of being far enough from the bounds of the variable.

Antigone is the most recent global MINLP solver in our list. Its performance on MINLP test problems is shown to be one of the best MINLP solvers with BARON. It can handle all classes of optimization problems and most types of functions are supported, including trigonometric functions.

3.3.6 Comparison

Concise computational experiments were made to test the solver performances in solving a few selected instances. These experiments are not extensive because the comparison between the solvers is not the main purpose of the dissertation. Twenty MINLP instances were selected from the MINLPLib library [1] and most of them are nonconvex. The instances were written by many authors from different sources, and they are distinctive in their structures and applications. The selection was on instances that are not trivial and not very difficult, so most of them were solved to global optimality by at least one solver.

An overview of the selected instances is given in Table 3.1. According to the library, instances number 8, 11, and 19 are not solved to optimality, but they are included in these experiments to test the solver performances in challenging instances. Instances that contain trigonometric functions are not selected because these functions are not supported by some of the solvers.

The computations were made using the solvers through the NEOS server [2] , and no changes were made on the solvers' default settings in the server. The time limit was set to 1000 seconds which was enough for the solvers to solve many instances to global optimality. Also, for instances that were not solved within the time limit, the solvers gave a sufficient insight into their performances on these instances. The comparison between the solver performances is in the time needed to find a solution, and on the quality of the solution (whether it was proven to be global or not).

The results of the computational experiments are summarized in Table 3.2. If a solver successfully finds the global optimal solution of an instance within the time limit, then the result will be written in the table as S, and the result will be F if

[1] `http://www.minlplib.org/index.html`

[2] `https://neos-server.org/neos/index.html`

Table 3.1: An Overview of the Instances, Where seq # is the Sequential Number of the Instance That Will Be Used Later as a Reference in the Results Table; # c, # d , and # cons Refer to the Number of Continuous, Discrete Variables, and Constraints; and obj val is the Best Found Value of the Objective Function. Instances with Asterisks at the End of Their Names are Convex.

Instance	seq #	# c	# d	# cons	obj val
autocorr_bern30-04	1	1	20	1	-324
batchs201210m*	2	307	251	2327	2295349
cecil_13	3	660	180	898	-115657
ex1252a	4	15	9	34	128894
fin2bb	5	414	175	618	0
gastrans135	6	961	232	2472	0
hda	7	709	13	718	-5964.53
heatexch_gen1	8	100	12	120	154896
jit1*	9	21	4	32	173983
johnall	10	4	190	192	-224.73
kport20	11	61	40	27	31.8093
minlphix	12	64	20	92	316.693
nvs20	13	11	5	8	230.922
oil2	14	934	2	926	-7.333
pooling_epa1	15	184	30	340	-280.806
procsel	16	7	3	7	-1.9231
sepasequ_convent	17	621	20	1128	482.5
sfacloc2_3_80	18	216	107	2268	11.0585
sporttournament20	19	1	190	1	192
waterno2_02	20	314	18	410	39.5714

the solver fails to produce any solution. If the time limit is reached (expressed as *TL* in the table), and the solver managed to find a local solution, then the result will be *L()* with the best objective value written between the parentheses. In case the solver found the global solution but failed to close the gap between the lower and upper bounds of the objective value (did not prove global optimality), then *G()* is written under the results column with the absolute gap between the parentheses. The absolute gap is the difference between the best objective value and its lower bound. The solver LINDOGlobal reached its size limit, which is denoted by *SL*, in few instances. Sometimes a solver finds a solution that is not the global one, but the solver claims the solution is global; in this case, the solving attempt is considered to be a failure.

From the results in Table 3.2, it can be seen that BARON has the most successful attempts, solving 16 out of 20 instances to global optimality. BARON also found the global solution (without closing the gap) for two instances, and for the instances for which BARON did not find their global solutions, they were not solved by any solver. SCIP and ANTIGONE also have good success rates solving 13 and 12 instances, respectively. ANTIGONE seemed to perform better than SCIP in finding the global solution (4 compared to 1) of the instances that were not solved within the time limit. With only 9 solved instances, LINDOGlobal has the least number of successes, despite the fact that it did not attempt to solve 3 instances due to the size limit.

To compare the speed between the solvers, only instances that were successfully solved by at least four solvers are considered. The average time is obtained by calculating the arithmetic mean of solving times in the considered instances. BARON needed an average of 7.58 seconds solving time per instance, which is the shortest time among the solvers. SCIP is the second fastest solver achieving an average of

Table 3.2: Computational Test Results.

	BARON		Couenne		LINDOGlobal		SCIP		ANTIGONE	
seq #	time	result	time	result	time	result	time	result	time	result
1	1.9	S	TL	G(88.7)	TL	G(12)	681	S	4.8	S
2	11	S	161	S		SL	4.2	S	2.4	S
3	0.8	S	42	S	6.5	S	0.4	S	1.1	S
4	7.7	S	9	S	85.5	S	14	S	TL	G(26585)
5	662	S	TL	L(349)	TL	L(0.006)	48	S	60	S
6	5.66	S	TL	F		SL	TL	F	667	F
7	4	F	TL	F	41.9	F	TL	L(621)	41	F
8	TL	G(54395)	TL	L(288241)	TL	L(160349)	14	L(685572)	TL	G(49461)
9	0.2	S	0.06	S	0.07	F	0.4	S	1.5	S
10	3.3	S	3	S	42.6	S	14	S	9.9	S
11	TL	L(32.23)	TL	L(32.31)	TL	L(32.21)	TL	L(32.12)	TL	L(32.13)
12	TL	G(∞)	0.88	S	10.9	S	TL	F	1.2	S
13	2.8	S	0.68	S	1.75	S	0.5	S	19	S
14	38	S	21.3	S	5.2	S	120	S	190	S
15	185	S	911	S	TL	G(176)	TL	L(-279.9)	293	S
16	0.3	S	0.01	S	0.02	S	0.03	S	0.01	S
17	20.1	S	72.4	F	38.1	S	24.2	S	0.21	F
18	127	S	TL	G(4.41)		SL	93.5	S	TL	G(0.5251)
19	75	S	TL	G(34)	0.06	F	TL	G(6.8)	tl	G(10.6)
20	4.1	S	14.3	F	46.7	S	9.4	S	47.7	S

18 seconds per instance, followed by LINDOGlobal, Couenne, and ANTIGONE with averages 27, 29, and 33 seconds, respectively.

Other interesting outcomes about the instances can be observed from the table. For example, although Instance 2 is one of the largest instances in size, it was solved easily by the solvers, confirming that convex problems are usually easy regardless of their size. Instance 19 can be a good example that not all quadratic problems can be handled easily, as the instance is quadratic but it was solved within the time limit only by BARON. All solvers failed to solve instances 7, 8, and 11 (8 and 11 are marked as unsolvable in the MINLPLib library), and many instances resulted in a failure by at least one solver. This outcome supports the fact that MINLP problems are still challenging, even with all the advances that have been made on the global solvers.

These results suggest that BARON performs better than other solvers in terms of speed and the number of handled instances, in spite of these experiments being brief. SCIP also is proved to be a reliable MINLP solver. More comprehensive experiments on instances from the MINLPLib library were done by Mittelmann (2020), and they suggested similar conclusion to the one that was discussed here.

Chapter 4

PIECEWISE LINEAR APROXIMATION

The rapid advances of linear optimization in the 20^{th} century led to the development of robust MILP solvers that can easily handle problems with millions of variables. Even with the current improvements in the MINLP solvers, it would be a great step forward if MINLP problems can be solved globally by MILP solvers. In principle, this can be done by approximating the MINLP problem by an MILP one, but solving this approximation by an MILP solver will probably be harder than solving the original problem by an MINLP solver.

The most common method to remodel MINLP as MILP is the *piecewise linear approximation* (PLA). This method takes an advantage of the fact that any continuous function can be approximated by a piecewise linear one.

Definition 4.1 *A function f is called a piecewise linear (PL) function if its domain can be partitioned into a finite number of subdomains such that f is linear over each subdomain.*

Replacing every nonlinear function in the MINLP model by their PLAs will yield an MILP model. Because of the size of the new approximated model, the PLA was not introduced to do full approximations to the MINLP models. Instead, it was used mostly to find linear under/over estimators to some of the functions involved in the models. In the following sections, PLA background will be provided, followed by a description of the most common PLA models. After that, a discussion on two approaches will be presented: one approach is on improving the quality PLA models,

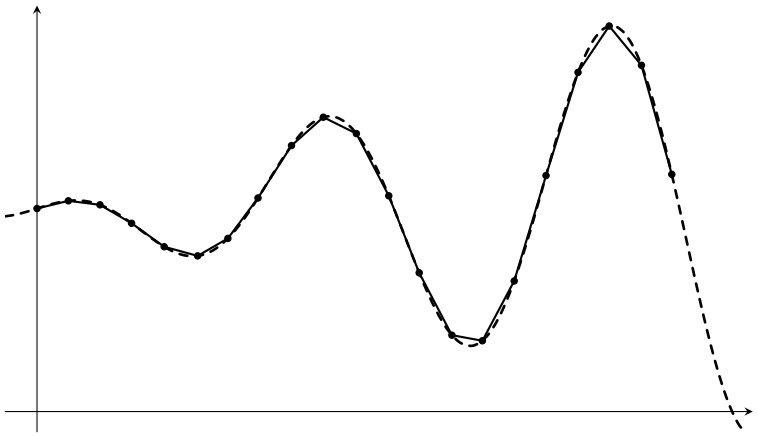


Figure 4.1: PLA of the Function $f(x) = 0.5x\cos(x) + 1$.

and the other approach will be on using them in a way that might improve the MINLP solvers.

4.1 Background

Modeling problems with nonconvex PL functions as MILP problems dates back to the 1950s, and many authors have discussed this topic since then (see, for example, Markowitz and Manne (1957), Dantzig (1960), and Jeroslow and Lowe (1984)). The procedures of approximating a nonlinear function $f(x)$, where $x \in [x_l, x_u] \subseteq \mathbb{R}$, by a PL function are simple. First, the domain of the variables x is divided into n intervals by introducing the *breakpoints* $x_0 = x_l < x_1 < x_2 < \cdots < x_n = x_u$ (Note that this notation is similar to the one that represents the vector of variables in the previous chapters, but since the functions in this chapter will not have more than two variables, there should not be confusion). Then the function f is evaluated at each breakpoint, and the lines that connect the points $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$ form the desired PL function, denoted by $\overline{f}$. The PLA idea was also extended to higher dimensional functions. An example of PLA of a one dimensional function is shown in Figure 4.1.

Increasing the number of breakpoints will increase the accuracy of the approximation, but it will result in problem size growth. This major drawback may restrict the

PLA benefits to functions of only few dimensions. Rebennack and Kallrath (2015) discuss minimizing the number of breakpoints needed to approximate a nonlinear function up to a given tolerance. This subject will not be discussed here. Doing PLA by introducing breakpoints, both binary and continuous variables must be introduced. The number of the new variables may increase exponentially with the number of dimensions of the function and it varies depending on the model used to do the PLA, as will be shown in the following sections.

4.2 One Dimensional PLA

It is important to mention that if a function with more than one variable is separable, then the one dimensional PLA is applicable to this function. Therefore, for the remainder of the discussion, any separable multidimensional function will be considered as one dimensional.

To do PLA of a function $f(x)$, the breakpoints $x_0 = x_l < x_1 < x_2 < \cdots < x_n = x_u$ for the variable $x \in [x_l, x_u] \subseteq \mathbb{R}$ are required by all PLA models that will be presented in this section. It is also required to introduce an ordered set of continuous variables and binary variables.

Definition 4.2 *A special ordered set of type k (SOSk) variables is a set of variables of which at most k variables can be nonzero, and all of the nonzero variables have to be adjacent.*

The idea of these variables was introduced by Beale and Tomlin (1970) as SOS (referring to SOS1) before other types of SOS variables were introduced. The tool of SOS is essential for most PLA models and can be generally useful in optimization.

A straight forward approximation to the function $f(x)$ is to replace it with $\overline{f} = \sum_{i=0}^{n} z_i x_i$, where the z_i's are SOS1 binary variables. This approximation will evaluate

the function f only at the breakpoints but not along the lines connecting the pairs $(x_i, f(x_i))$. Therefore, more accurate approximation can be obtained by the models that will be described below. These models require introducing continuous SOS2 variables in addition to binary variables and linear constraints.

4.2.1 Convex Combination

The *convex combination model* (also called the λ-model in some sources) is one of the most common PLA models and it was introduced by Dantzig (1960). As suggested by the name, this approximation replaces the nonlinear function by a convex combination between the function values of two adjacent breakpoints, and the variable x is expressed as convex combination of these breakpoints. This model requires introducing SOS2 continuous variables, with each variable corresponding to one breakpoint, and these variables are denoted by λ_i, where $\lambda_i \in [0, 1]$ for $i = 0, 1, \ldots, n$. To ensure that the λ_i's are SOS2, the binary variables z_i, for $i = 1, 2, \ldots, n$ are required. The binary variables correspond to the intervals between the breakpoints. e.g., a binary variable z_i corresponds to the interval $[x_{i-1}, x_i]$ and the variable x can take value from this interval only if $z_i = 1$. The PLA of the nonlinear function f using the convex combination model will be as follows:

$$f = \sum_{i=0}^{n} \lambda_i f(x_i) \tag{4.1a}$$

$$x = \sum_{i=0}^{n} \lambda_i x_i \tag{4.1b}$$

$$\lambda_0 \leq z_1 \tag{4.1c}$$

$$\lambda_n \leq z_n \tag{4.1d}$$

$$\lambda_i \leq z_i + z_{i+1} \quad \text{, for } i = 1, \ldots, n-1 \tag{4.1e}$$

$$\sum_{i=1}^{n} z_i = 1 \tag{4.1f}$$

$$\sum_{i=0}^{n} \lambda_i = 1. \tag{4.1g}$$

The right hand side of Equation 4.1a would replace the function f in the original optimization problem and the rest of the equations are added as constraints. Since Equation 4.1f guarantees that exactly one binary variable will take the value one, Constraints 4.1c to 4.1e ensure that no more than two λ's can be greater than zero. It can be concluded from Equation 4.1g that the sum of the two nonzero λ's is one, enforcing the convex combination condition.

Replacing the nonlinear function f by the piecewise linear $\overline{f}$ will come at the price of adding n binary variables, $n+1$ continuous variables, and $n+5$ constraints. These numbers are not too large and can be handled easily by MILP solvers, but if many nonlinear functions are present in an optimization problem and they need many breakpoints to get a good approximation, then MILP solvers might not be able to handle the resulting size of the problem.

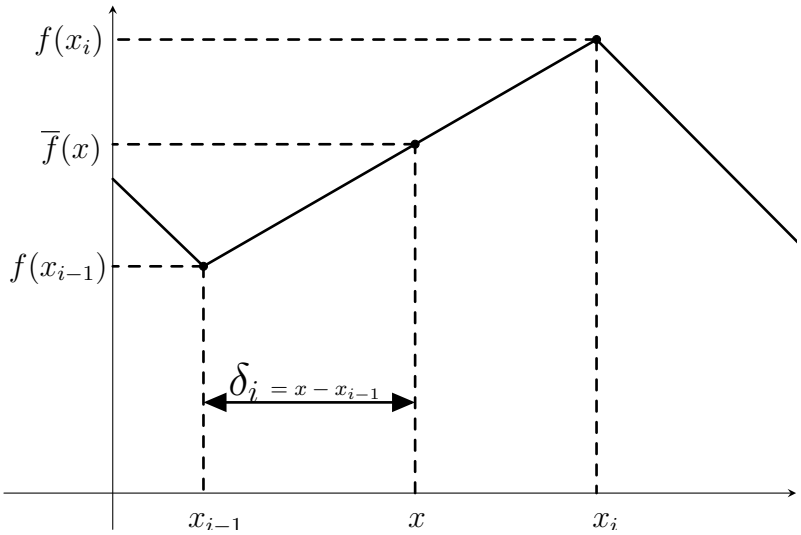


Figure 4.2: The Incremental Method.

4.2.2 Incremental Method

Another model that was introduced by Markowitz and Manne (1957) is the *incremental model*. It is also known as the δ-model because of the notation that is used to represent the introduced variables. The positive continuous variables that are introduced in this model are δ_i and they correspond to the intervals $[x_{i-1}, x_i]$, for $i = 1, \ldots, n$, instead of the breakpoints as in the convex combination model. The variable x that is taking a value in interval j can be expressed as $x = x_{j-1} + \delta_j$. As shown in Figure 4.2, the PL function value at that point can take the form $\overline{f}(x) = f(x_{i-1}) + s_i \delta_i$, where s_i is the slope of the i^{th} segment of $\overline{f}$ which is given by

$$\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}.$$

After introducing the binary variables z_i, for $i = 1, \ldots, n-1$, the incremental model will be:

$$f = f(x_0) + \sum_{i=1}^{n} s_i \delta_i \tag{4.2a}$$

$$x = x_0 + \sum_{i=1}^{n} \delta_i \tag{4.2b}$$

$$(x_i - x_{i-1}) z_i \leq \delta_i \quad \text{, for } i = 1, \ldots, n-1 \tag{4.2c}$$

$$\delta_{i+1} \leq (x_{i+1} - x_i) z_i \quad \text{, for } i = 1, \ldots, n-1. \tag{4.2d}$$

For Equation 4.2b to be feasible, $\delta_i = x_i - x_{i-1}$ must hold whenever $\delta_{i+1} > 0$. This condition is satisfied by Constraints 4.2c and 4.2d, and they also ensure that if $z_i = 0$, then $z_j = 0$ and δ_{j+1} for all $j > i$, and if $z_i = 1$, then $z_j = 1$ for all $j < i$. To illustrate this, assume $z_j = 1$, then $x_j - x_{j-1} \leq \delta_j$ follows from 4.2c, and the inequality $\delta_j \leq (x_j - x_j) z_{j-1}$ will follow from 4.2d. Now if $z_{j-1} = 0$ then there will be a contradiction between the two inequalities, so z_{j-1} must take the value 1.

It can be seen that the incremental model has one less binary variable and one less continuous variable, but it has nearly twice as many constraints, compared to the convex combination model.

4.2.3 Multiple Choice Method

The incremental model was modified by Balakrishnan and Graves (1989), leading to the *multiple choice model*. The value of any variable δ_i in the incremental model must be in the interval $[0, x_i - x_{i-1}]$, but in this model, the variable δ_i can either be 0 or take its value from the interval $[x_{i-1}, x_i]$. Thus, if the value of variable x is in the interval $[x_{i-1}, x_i]$ then $x = \delta_i$. The function value at x can be computed using the slope and the *y-intercept* of segment i, denoted by f_i, and it can be calculated by $\overline{f}(x) = s_i \delta_i + f_i$. an illustration of these notations is given in Figure 4.3.

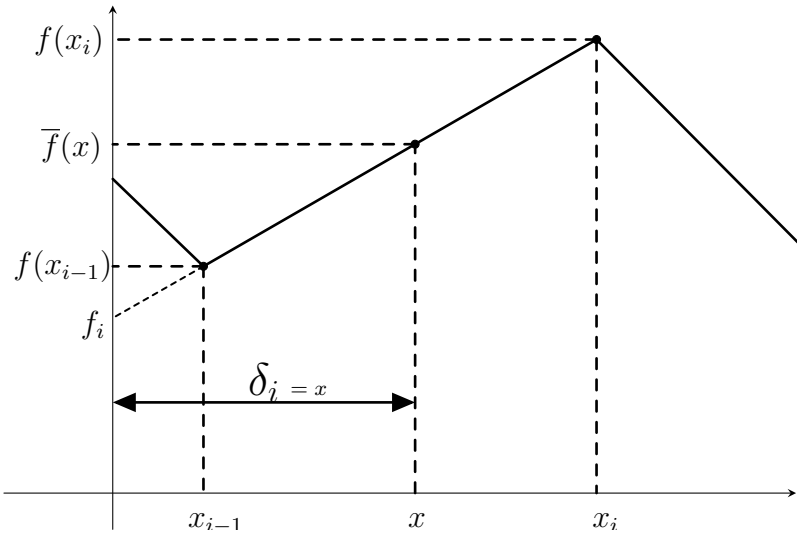


Figure 4.3: The Multiple Choice Method.

To ensure that only one of the δ_i's can take a nonzero value, a constraint that allows only one binary variable to be nonzero must be added. The multiple choice model will take the form:

$$f = \sum_{i=1}^{n} s_i \delta_i + f_i z_i \tag{4.3a}$$

$$x = \sum_{i=1}^{n} \delta_i \tag{4.3b}$$

$$x_{i-1} z_i \leq \delta_i \quad \text{, for } i = 1, \ldots, n \tag{4.3c}$$

$$\delta_i \leq x_i z_i \quad \text{, for } i = 1, \ldots, n \tag{4.3d}$$

$$\sum_{i=1}^{n} z_i = 1. \tag{4.3e}$$

Equation 4.3e along with Constraints 4.3c and 4.3d indicate that if, for example, $z_j = 1$, then $x_{j-1} \leq \delta_j \leq x_j$ and $\delta_i = 0$ for any $i \neq j$, therefore, the feasibility of Equations 4.3a and 4.3b is guaranteed. The number of newly introduced variables and constraints is similar to the incremental model.

4.2.4 Disaggregated Convex Combination

Another PLA model is the *disaggregated convex combination model* which uses the same idea as the convex combination model. The only difference is that instead of introducing one continuous variable corresponding to each breakpoint, two positive variables are introduced for each interval. These two variables will be associated with the lower bound of the interval and the upper bound, denoted by λ_i^l and λ_i^u, for $i = 1, \ldots, n$, resulting in the following model:

$$f = \sum_{i=1}^{n} \lambda_i^l f(x_{i-1}) + \lambda_i^u f(x_i) \tag{4.4a}$$

$$x = \sum_{i=1}^{n} \lambda_i^l x_{i-1} + \lambda_i^u x_i \tag{4.4b}$$

$$\lambda_1^u \leq z_1 \tag{4.4c}$$

$$\lambda_n^l \leq z_n \tag{4.4d}$$

$$\lambda_i^l + \lambda_i^u \leq z_i \quad \text{, for } i = 1, \ldots, n \tag{4.4e}$$

$$\sum_{i=1}^{n} z_i = 1 \tag{4.4f}$$

$$\sum_{i=1}^{n} \lambda_i^l + \lambda_i^u = 1. \tag{4.4g}$$

Even though the disaggregated convex combination model requires almost double the number of continuous variables needed by the convex combination model, it will have a *tighter* approximation; that is the integrality constraints of the new MILP model are satisfied by the extreme points of its LP relaxation. More about this model can be found in Meyer (1976); Jeroslow and Lowe (1984).

4.2.5 Logarithmic Formulation

As the number of the breakpoints increases, the more accurate PLA can be obtained, and for all of the models mentioned so far, the required number of binary variables is almost the same as the number of breakpoints. A large number of binary variables may lead to a deep and unbalanced B&B tree if the solver's branching priority is on the binary variables, which is the case for most solvers. As a result, the efficiency of MILP solvers will be affected if the PLA is done using many breakpoints, especially for higher dimensional functions. On the other hand, if the number of breakpoints is reduced, the solvers might handle the PLA well, but the solution probably will not be good enough to the original problem.

Many attempts have been made to deal with the branching problem such as the one suggested by Geißler *et al.* (2012), which modifies the branching strategies mentioned by Beale and Tomlin (1970); Beale and Forrest (1976) and others. The modification basically forces the branching to be on the continuous variables only. However, these attempts do not resolve the size problem completely. Vielma *et al.* (2010); Vielma and Nemhauser (2011) introduced a technique that allows PLA models to use dramatically fewer binary variables. They applied the technique to both versions of the convex combination model and denoted it by *logarithmic model*, and it was later applied to other PLA models. In this section, it will be shown how to use the logarithmic formulation in the convex combination model.

The logarithmic convex combination model with $n+1$ breakpoints requires introducing only $\lceil \log_2 n \rceil$ binary variables, denoted by z_i, instead of n (required by the convex combination model). To do this, every interval is assigned a binary vector in $0,1^{\lceil \log_2 n \rceil}$ using the so-called Gray code. Therefore, the assigned vectors of any two adjacent intervals will differ only by one component.

Example 4.1 *Assume PLA is done using the convex combination model with five intervals. The number of needed binary variables will be* $\lceil \log_2 5 \rceil = 3$. *A possible set of assigned vectors will be* $000, 001, 011, 010, 110$ *in this particular order.*

After assigning suitable vectors for the intervals, two index sets for every $k \in \{1, 2, \ldots, \lceil \log_2 n \rceil\}$, denoted by $I^1(k)$ and $I^0(k)$, are introduced. Let $I^1(k)$ be the set of breakpoint indices where every binary vector of the breakpoint's adjoining intervals has one at its k^{th} component; and $I^0(k)$ will be the index set where the binary vector has zero at its k^{th} component. Back to Example 4.1, the index sets are given as follows:

$$I^1(1) = 5 \qquad I^0(1) = 0, 1, 2, 3$$
$$I^1(2) = 3, 4, 5 \qquad I^0(2) = 0, 1$$
$$I^1(3) = 2 \qquad I^0(3) = 0, 4, 5.$$

With the index sets defined, the logarithmic convex combination model can be expressed by:

$$f = \sum_{i=0}^{n} \lambda_i f(x_i) \tag{4.5a}$$
$$x = \sum_{i=0}^{n} \lambda_i x_i \tag{4.5b}$$
$$\sum_{i \in I^1(k)} \lambda_i \leq z_k \qquad \text{, for } k = 1, \ldots, \lceil \log_2 n \rceil \tag{4.5c}$$
$$\sum_{i \in I^0(k)} \lambda_i \leq 1 - z_k \qquad \text{, for } k = 1, \ldots, \lceil \log_2 n \rceil \tag{4.5d}$$
$$\sum_{i=0}^{n} \lambda_i = 1. \tag{4.5e}$$

In this model, the SOS2 condition on the continuous variables is enforced by Constraints 4.5c and 4.5d. The two variables that are allowed to be nonzero are the ones associated with the interval whose binary vector is $(z_1, z_2, \ldots, z_{\lceil \log_2 n \rceil})$.

4.2.6 Comparison

To compare between these PLA models, some of their theoretical and numerical properties will be studied. Croxton *et al.* (2003) presented a good overview of the models mentioned in the previous sections and showed they are equivalent in terms of the feasibility of their solutions. In term of tightness, it is desired for a PLA model to be *locally ideal*. According to Padberg (2000), a model is locally ideal if the vertices of its LP relaxation satisfy the integrality constraints of the original problem. Vielma *et al.* (2010) proves that all PLA models mentioned earlier are locally ideal except the convex combination model. However, this model has the *sharpness* property, i.e,. the projection of the vertices of the model onto the original set of the variables is exactly the convex hull of the set. Also it can be shown that any locally ideal model is sharp. More recent theoretical comparison between the models was given by Sridhar *et al.* (2013).

In theory, all models have similar properties, but in terms of the number of newly introduced variables and constraints, they are different. It is clear that the logarithmic formulation of the models will have the advantage of requiring considerably fewer binary variables. In practice, it was shown that logarithmic models have similar performance to other models for few breakpoints , but they have the advantage over other models when a larger number of breakpoints is used or higher dimensional functions are approximated.

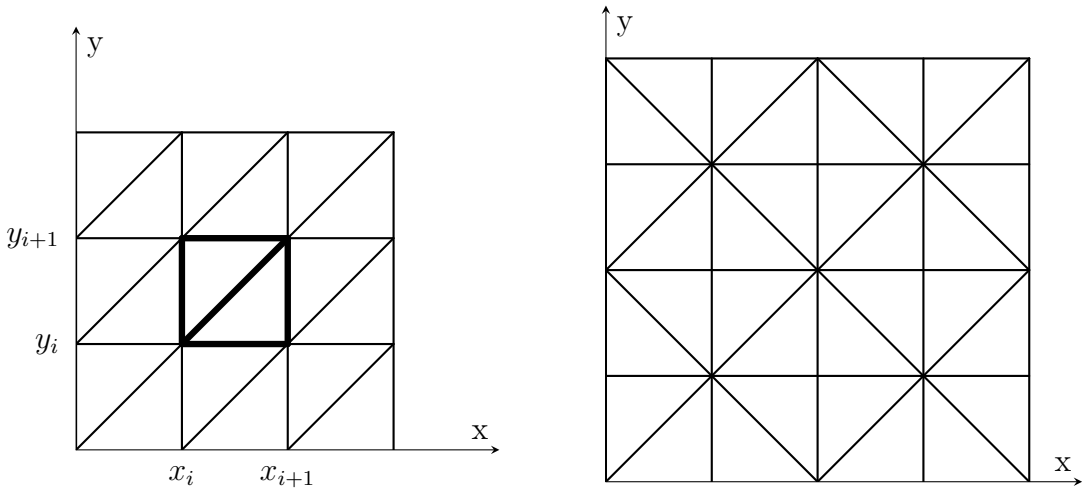
In the computational experiments made by Vielma *et al.* (2010), they approximated functions with one variable in 100 test instances, and used CPLEX to solve

the PLAs. It was observed that for less than 10 breakpoints, all models performed well with the multiple choice model being slightly better. As the number of breakpoints increases, the logarithmic models started to gain the upper hand. When 33 breakpoints were used, the logarithmic models are around 20 times faster than the incremental model, which is more than two times faster than the convex combination and multiple choice models. The disaggregated convex combination is much slower than the rest. Similar outcomes resulted from testing 100 problems where the approximated functions have two variables. A less extensive experiment was done by Geißler *et al.* (2012), and it was concluded that in some cases, it is better to use the incremental model rather than the logarithmic one, even though the latter has smaller size.

4.3 Two Dimensional PLA

All of the models presented in the previous section can be extended to higher dimensions. A discussion about the PLA of general n-dimensional functions can be found in Geißler *et al.* (2012) and Vielma *et al.* (2010). In this section, PLA of functions with two variables will be discussed.

Similarly to the one dimensional models, the domains of x and y are divided into n and m intervals by the breakpoints $x_0 = x_l < x_1 < x_2 < \cdots < x_n = x_u$ and $y_0 = y_l < y_1 < y_2 < \cdots < y_m = y_u$, respectively. For every two adjacent breakpoints x_i and x_{i+1} along the x-axis, and y_j and y_{j+1} along the y-axis, two triangles can be formed: lower and upper triangle as illustrated in Figure 4.4a. This partitioning of the domain is called a *triangulation*, and it can be done in different ways. Then the function $f(x, y)$ is evaluated at each vertex to approximate it by the PL function formed by simplices instead of lines, as in the one dimensional case.



(a) A Valid Triangulation.

(b) Union Jack Triangulation.

Figure 4.4: Different Triangulations of the Variables' Partitioned Domain.

In this section, the convex combination model will be extended to approximate a function of two variables $f(x, y)$. The extension to other models will not be discussed here, but a study of two dimensional PLA can be found in D'Ambrosio *et al.* (2010), where three models are explained and a theoretical comparison, as well as computational tests, are presented.

Also the 2d logarithmic formulation will be discussed in this section. According to Vielma *et al.* (2010), the triangulation needed for logarithmic convex combination is required to be equivalent to the *Union Jack* triangulation, illustrated in Figure 4.4b. Since the logarithmic disaggregated convex combination model can accept any triangulation, it will be presented later, and for the remainder of the dissertation, only the triangulation in Figure 4.4a will be considered.

4.3.1 Convex Combination Model

For the two dimensional convex combination model, to express the point (x, y) and the function $f(x, y)$ as convex combination of vertices and the function values at these

vertices, the continuous variables $\lambda_{ij} \in [0,1]$, for $i = 0, 1, \ldots, n$ and $j = 0, 1, \ldots, m$, are introduced and associated with the vertices. Since only the variables associated with the vertices of one triangle are allowed to be nonzero, they need to be introduced as SOS3 variables.

The SOS3 condition is ensured by introducing the binary variables z^u_{ij} and z^l_{ij} associated with the upper and lower triangle of the rectangle formed by the intervals $[x_{i-1}, x_i]$ and $[y_{j-i}, y_j]$. The PLA model will be:

$$f = \sum_{i=0}^{n} \sum_{j=0}^{m} \lambda_{ij} f(x_i, y_j) \tag{4.6a}$$

$$x = \sum_{i=0}^{n} \sum_{j=0}^{m} \lambda_{ij} x_i \tag{4.6b}$$

$$y = \sum_{i=0}^{n} \sum_{j=0}^{m} \lambda_{ij} y_j \tag{4.6c}$$

$$\lambda_{i-1,j-1} \leq z^u_{ij} + z^l_{ij} + z^u_{i,j-1} + z^l_{i-1,j-1} + z^u_{i-1,j-1} + z^l_{i-1,j}$$
$$\text{for } i = 1, \ldots, n-1 \text{ and } j = 1, \ldots, m \tag{4.6d}$$

$$\sum_{i=0}^{n} \sum_{j=0}^{m} z^u_{ij} + z^l_{ij} = 1 \tag{4.6e}$$

$$\sum_{i=0}^{n} \sum_{j=0}^{m} \lambda_{ij} = 1, \tag{4.6f}$$

with the dummy variables $z^*_{0*} = z^*_{*0} = 0$. Equation 4.6e guarantees that exactly one triangle is picked, where Constraint 4.6d ensures that only the variables corresponding to the vertices of that triangle can be nonzero. Therefore, the SOS3 condition is satisfied. The number of needed binary variables is $2nm$, in addition to nm continuous variables and nm constraints, so using many breakpoints would result in a huge PL problem. Many computational tests, including ours, showed that using more than 10 breakpoints will produce complicated approximation, and with

10 or less, the approximation will be very inaccurate if the problems have large variable domains. Thus, using a logarithmic formulation to reduce the number of binary variables can resolve the size issue.

4.3.2 Logarithmic Disaggregated Convex Combination Model

The 2d logarithmic formulation is similar to that in one dimension. In this case, every simplex is assigned a binary vector with $\lceil \log_2 2nm \rceil$ components, assuming the triangulation in Figure 4.4a is used. Let $\mathcal{S} = \{s_1, s_2, \ldots, s_{2nm}\}$ be the set of all simplices in the triangulation. For every simplex $s \in \mathcal{S}$, three positive continuous variables are introduced, and let Λ be the set of all variables. Note that the common vertex shared by multiple simplices, will be associated with multiple variables. For any variable $\lambda \in \Lambda$, define the point (x_λ, y_λ) to be the vertex associated with this λ. Finally, for $k \in \{1, 2, \ldots, \lceil \log_2 2nm \rceil\}$, let $I^1(k) \subset \Lambda$ and $I^0(k) \subset \Lambda$ be the sets of variables associated with simplices whose binary vector has one and zero, respectively, on its k^{th} component. The model now can be described as:

$$f = \sum_{\lambda \in \Lambda} \lambda f(x_\lambda, y_\lambda) \tag{4.7a}$$

$$x = \sum_{\lambda \in \Lambda} \lambda x_\lambda \tag{4.7b}$$

$$y = \sum_{\lambda \in \Lambda} \lambda y_\lambda \tag{4.7c}$$

$$\sum_{\lambda \in I^1(k)} \lambda \leq z_k \qquad \text{, for } k = 1, \ldots, \lceil \log_2 2nm \rceil \tag{4.7d}$$

$$\sum_{\lambda \in I^0(k)} \lambda \leq 1 - z_k \qquad \text{, for } k = 1, \ldots, \lceil \log_2 2nm \rceil \tag{4.7e}$$

$$\sum_{\lambda \in \Lambda} \lambda = 1. \tag{4.7f}$$

Constraints 4.7d and 4.7e ensure that only the variables corresponding to the simplex with binary vector $(z_1, z_2, \ldots, z_{\lceil \log_2 2nm \rceil})$ are allowed to be nonzero, and with Constraint 4.7f, the convex combination condition is satisfied. It can be observed that this model requires many fewer binary variables and constraints than the 2d convex combination model. On the other hand, the number of continuous variables is six times larger in the logarithmic model. Comparing the two models described in this section, as will be shown in Chapter 5, the logarithmic model performs better.

4.4 Improving PLA Models

In this section, a few approaches to improve the PLA models will be presented. Since PLA requires introducing many binary and continuous variables and constraints, it was not studied as a stand-alone method to solve MINLP problems until recently. It is usually used as a tool in optimization algorithms to find local solutions or to under/over estimate some of the nonlinear functions. For PLA models to completely transform an already hard MINLP problem and produce an MILP problem that is easier than the original to be handled, more improvements on these models are needed. Note that even if the targeted problems in this dissertation have many variables, the PLA models will be applied separately only to functions of two variables or less.

One approach to improve the models is introducing a new method to choose the breakpoints. This approach was motivated by the desire to produce an accurate approximation with reasonable problem size. Unfortunately, this is rarely possible since an accurate approximation requires many breakpoints. One of the methods to overcome this issue is to study how to partition a domain. Most existing PLA models choose the locations of breakpoints based on a uniform partitioning of the domain. The approach proposed in this chapter, namely a nonuniform partitioning, leads to

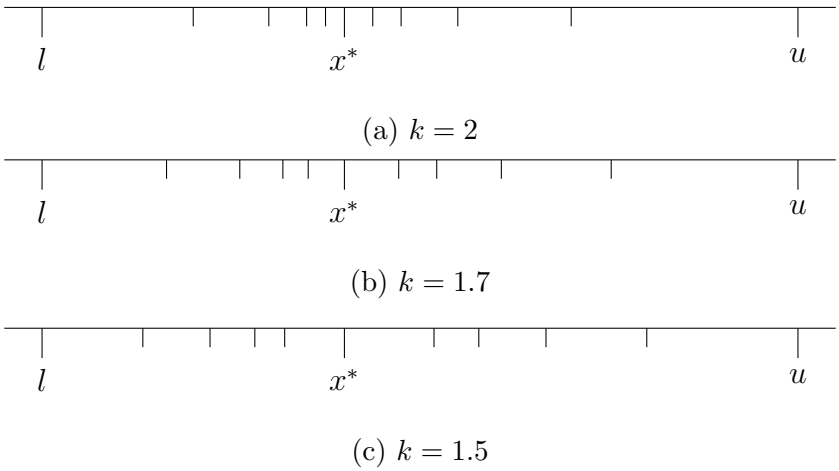
a better PLA model than the one produced by uniform partitioning with the same number of breakpoints. The details are given in Section 4.4.1

Another approach that will be presented in this chapter is applying the PLA partially to problems with many nonlinear functions. Given a complicated MINLP problem, applying the PLA to only some of its nonlinear constraints will not make the problem solvable by an MILP solver, but it might make the problem less complicated for MINLP solvers. The idea is to identify complicated nonlinear constraints and approximate some of them by linear ones, then the modified problem is solved using MINLP solvers. An algorithm that identifies these constraints and approximates them will be provided in Section 4.4.2.

4.4.1 New Method to Choose BreakPoints

Most PLA models, that are used within the context of optimization, rely on uniform partitioning of variable domains. A few methods to do nonuniform partitioning were introduced within other contexts. For example, Dahl and Realfsen (1996) used nonuniform partitioning to approximate a one dimensional curve. The idea is to add more breakpoints in the part of the domain where the function has higher curvature. This method is done by solving a shortest path problem, and it was later used within the PLA approach for one and two dimension by Vasudeva (2015). Other methods were introduced to use nonuniform partitioning to get piecewise convex/linear relaxations, such as the one proposed by Nagarajan *et al.* (2019).

The partitioning method suggested in this section is to build the partitioning around a local solution. The density of breakpoints increases as they get closer from both sides to the local solution, which itself is a breakpoint. Assume the local solution of a variable $l \leq x \leq u$ is x^*, then a possible partitioning is to have a breakpoint in the halfway in the interval from the upper/lower bound to x^*, then another point



(a) $k = 2$

(b) $k = 1.7$

(c) $k = 1.5$

Figure 4.5: Partitioning Using the Formulas $x^* + \frac{u-x^*}{k^i}$ and $x^* - \frac{x^*-l}{k^i}$.

halfway toward x^* and so on. This partitioning can be given by using the formulas

$$x^* + \frac{u - x^*}{k^i} \qquad \text{and} \qquad x^* - \frac{x^* - l}{k^i}, \tag{4.8}$$

for $k = 2$ and $i = 1, 2, \ldots,$ to get the breakpoint values on both sides of $x*$, as shown in Figure 4.5a.

The problem with the case $k = 2$ is that it leaves half of the interval between the upper/lower bound and x^* without any breakpoints, which may affect the accuracy of the PLA. Thus, giving k different values between one and two, as in Figures 4.5b and 4.5c, may yield better partitioning. It can be noticed that as k gets closer to one, the density of breakpoints shifts away from $x*$. If x^* happened to be at or very close to one of the bounds, the partitioning will be on one side only. In the case that the approximated function has two variables, the same logic is applied to both domains.

The PLA using this partitioning was tested for many values of $1 < k < 2$ against PLA with uniform partitioning. The results show that models with nonuniform partitioning were solved faster and yielded better solutions to most of the tested instances. Details about the targeted optimization problems and the test results will be given in Chapter 5.

4.4.2 Partial PLA

In this section, PLA will be performed only on parts of a given MINLP problem, instead of approximating all nonlinear functions. This is done by targeting functions with high nonlinearity to be handled by PLA, leaving the remaining functions unchanged. The goal of this approach is to avoid introducing unnecessary variables that result from approximating simple functions. To test this approach, an algorithm is introduced to deal with problems having many nonlinear constraints by picking one constraint at every iteration and approximating it, until enough constraints are replaced.

Assume the algorithm is applied to an MINLP problem of the form 2.1 with x^* and f^* as its global optimal solution and objective function value, respectively. The algorithm starts by solving the problem using an MINLP solver for a few nodes before it stops the solving process when a specified number of nodes, N, is reached. Then all nonlinear constraints are identified, and since the solving process was interrupted, using the current node solution will probably result in some of these constraints being violated. Now all nonviolated constraint are considered to be easy since the solver was able to satisfy them within few nodes, so the PLA will not be applied to these constraints.

Now the algorithm picks a constraint from the violated ones to be approximated using a PLA model, and this constraint is suggested to be the most violated one. As a result, the problem now has one less nonlinear constraint. Then the process is repeated until the problem has no violated constraints. At this point, the problem is solved regularly to produce a solution $\overline{x}^*$ to the modified problem with a function value $\overline{f}^*$. $|f^* - \overline{f^*}|$ is evaluated and the solving times of the original and modified

problems are compared. Algorithm 1 shows the steps of the partial PLA approach, for some $\epsilon > 0$ and $k \in \mathbb{N}$.

Obviously, if k is large enough, then all violated constraints will be replaced and the resulting problem will probably be harder to solve because of the size. The computational experiments with large k came as expected and produced extremely large problems. Then k was set to one, i.e., PLA was applied to only the most violated constraint in the first iteration. For some test instances, both PLA models, that were introduced in the previous section, produced modified problems that were solved faster than the original ones, with similar solution and objective value. As the number of replaced constraints increases, the modified problem gets more complicated. Most of the tested instances appeared to be better off without PLA if $k \geq 3$. In Chapter 5, a summary of the test instances will be presented in addition to the main findings on applying PLA to only parts of these instances.

Algorithm 1: An algorithm that chooses some constraints to be approximated.

Input:MIQCP problem;

Output:MIQCP problem with up to k constraints being approximated;

while $|\overline{f^*} - f^*| \geq \epsilon$ *and iteration* $\leq k$ **do**

  start the solving process;

  if *solving is done before number of nodes reaches* n **then**

    set the current objective value = $\overline{f^*}$;

    if $|\overline{f^*} - f^*| \geq \epsilon$ **then**

      add more breakpoints;

    end

  else

    stop the solving process when the tree has N nodes;

    identify the quadratic constraints;

    if *number of violated constraints* ≥ 1 **then**

      replace the most violated one with its PLA;

      set $\overline{f^*} = \infty$;

    else

      solve the PL problem and set the current objective value = $\overline{f^*}$;

    end

  end

end

Chapter 5

COMPUTATIONAL EXPERIMENTS

This chapter presents detailed reports on computational experiments that target QCP and MIQCP problems. The approaches that were introduced in the previous chapter can be applied to MINLP problems, but they were implemented to test the quadratic problems only. Most of these problems are not trivial and are based on different real world applications. The QCQP problems were taken from the *kall* instances in the MINLPLib library, whereas the MIQCQP problems were taken from both MINLPLib and QPLIB [1] . Two dimensional functions are involved in all of the instances, so 2d PLA is required.

The non linear functions in the instances are approximated by linear ones using the convex combination and logarithmic disaggregated convex combination models described by 4.6 and 4.7, and for the remainder of this chapter, they are referred to by CC and LOG, respectively. If the breakpoints needed for CC and LOG are determined using nonuniform partitioning, then the models will be called NCC and NLOG. These four models are also used to do partial PLA to the test instances by approximating all the nonlinear functions in one constraint.

The PLA models were implemented using PySCIPOpt [2] , which is a Python interface for the global solver SCIP. For now, the written codes can only approximate quadratic and bilinear terms. After the instance is read by SCIP, all quadratic constraints are identified before the domain of every variable involved in a quadratic function is partitioned (uniformly or nonuniformly). Then SCIP adds all necessary vari-

[1] `http://qplib.zib.de/`

[2] `http://scip-interfaces.github.io/PySCIPOpt/docs/html/index.html`

ables and constraints needed to replace the targeted function. Finally, any quadratic constraint is deleted from the instance and linear constraints, that approximate the deleted ones, are added. Now the instance is written in a format that can be read by MILP solvers.

The local instance solutions, that are needed for the nonuniform partitioning, are generated using the solver Knitro. The comparison between the uniform and nonuniform partitioning is determined through solving the MILP problems resulting from applying CC, LOG, NCC, and NLOG to the test instances. The MILP problems are solved using CPLEX 12.10 with its default settings, and the computations are done in a Linux machine with Intel Xeon E5-2620 2GHz processor. The time limit for each problem is set to one hour.

The partial PLA tests are done by solving the original instances and the same instances with one constraint replaced. The instances are solved using SCIP Optimization Suite 6.0.0 on a Linux machine with Intel Core i5-7y54 1.2GHz processor. The default solver settings were used with a time limit of two hours.

The results of the computational experiments will be described in detail in the following sections. Section 5.1 will show the results regarding the continuous instances, QCQP, while in Section 5.2, the findings of MIQCQP instance tests will be shown. Both sections will provide instance statistics and compare between the CC and LOG models with respect to the sizes of MILP problems produced by them. Also the sections will discuss the computational test results, where solving speed and approximation quality will be examined. At the end of the chapter, Section 5.3

will summarize the outcome of solving some examples of MINLP problems with two variables using the PLA models.

5.1 Continuous Variables

The computational results that are reported in this section are for the kall instances, where the objective is linear and the constraints are linear and quadratic. The quadratic constraints contain up to two functions with two variables of the forms $(x + y)^2$ or xy. Also, one variable functions of the form x^2 appear in the constraints of some of the problems.

Statistics of the instances are given in Table 5.1, where seq# is the sequential number that will refer to the corresponding instance throughout the section. The number of variables is shown under #v, and #cons (q) represents the total number of constraints including (q) quadratic ones. The best objective values found so far (according to the MINLPLib library) are entered under the obj val column header. All objective values shown in the table are proven to be the global optimum by at least 3 global solvers except instances 2, 3, 5, and 12.

The instances were solved regularly by SCIP, with time limit of 2 hours. The solving times are recorded and presented in seconds in the last column of Table 5.1. For some instances, the limits, machine memory (ML) or time (TL), were reached before the instance is solved to optimality. Also it can be observed from the solving times that some instances are trivial and solved quickly but others took time to be solved. Solving the instances here was not intended to find their global solutions, since the best solutions are already found and listed in the library and no further improvement to the solutions can be done. Instead, they were solved to give an idea about their difficulty levels, and to have a reference when it is needed to compare their results to the modified problems results.

Table 5.1: Statistics of Continuous Test Instances, and the Solving Time by SCIP.

instance	seq#	#v	#cons (q)	obj val	time
kall_circles_c6a	1	18	54 (22)	2.1117	1927
kall_circles_c6b	2	18	54 (22)	1.9736	2964
kall_circles_c7a	3	20	69 (29)	2.6628	2612
kall_circles_c8a	4	22	86 (37)	2.5409	TL
kall_congruentcircles_c31	5	10	16 (4)	0.6438	1
kall_congruentcircles_c32	6	10	16 (4)	1.3759	1
kall_congruentcircles_c41	7	12	24 (6)	0.8584	1
kall_congruentcircles_c42	8	12	24 (7)	0.8584	1
kall_congruentcircles_c51	9	14	34 (11)	1.073	12
kall_congruentcircles_c52	10	14	34 (11)	1.5371	4
kall_congruentcircles_c62	11	16	46 (16)	1.2876	16
kall_congruentcircles_c63	12	16	46 (16)	1.2876	11
kall_congruentcircles_c72	13	18	60 (22)	1.9663	225
kall_diffcircles_10	14	24	71 (45)	11.9355	6356 (ML)
kall_diffcircles_5a	15	14	24 (11)	5.1162	63
kall_diffcircles_5b	16	14	24 (11)	5.1162	44
kall_diffcircles_6	17	16	31 (16)	7.7879	102
kall_diffcircles_7	18	18	40 (22)	7.1531	177
kall_diffcircles_8	19	20	49 (28)	14.4813	3350
kall_diffcircles_9	20	22	60 (36)	13.3503	5118 (ML)

Before discussing the results of instance PLAs, one important factor about the instances has to be taken into account: the sizes of variable domains. When the variable involved in PLA of a function has a large domain, the domain will need more breakpoints to get a good approximation. However, introducing many breakpoints will affect the size, so the number of breakpoints for the computational tests is set to 10 for all domains. The domain size for each variable in the tested instances ranges between 1 and 18, with average size of 7.25 per domain. Therefore, 10 points should be enough to get good models for test purposes.

To compare between the PLA models with respect to the sizes of the produced problems, the instances were transformed into MILP problems by the models CC and LOG (using nonuniform partitioning would give the same size). Each instance was approximated by these models using 10, 20, and 30 breakpoints. Then the average numbers of constraints and binary/continuous variables per problem is recorded for each set of breakpoints, as shown in Table 5.2. It is apparent from the table that there is a significant advantage for the model LOG in terms of needed number of binary variables and constraints. Moreover, it will be shown later that this advantage is not outweighed by the CC model's advantage of having many fewer continuous variables.

Although Table 5.2 gives enough insight on the problem seizes, detailed statistics for every problem approximated using 10 breakpoints is given in Table 5.3. This table makes it easier to track the detail of every tested problem and compare between its size and computational results. It should be expected that most problems will not be solved easily by CPLEX due to the large number of binaries and constraints.

5.1.1 Uniform vs Nonuniform Partitioning

The computational experiments on solving the MILP problems approximating the selected instances were made by CPLEX with a time limit of one hour per problem.

Table 5.2: Average Sizes per Problem Produced by CC and LOG Models Using 10, 20, and 30 Breakpoints.

		10	20	30
BV	CC	6135	27195	63355
	LOG	303	377	414
CV	CC	3808	15083	33916
	LOG	22738	90416	203417
Cons	CC	3983	15259	34092
	LOG	764	891	983

The objective of these experiments is to compare between uniform and nonuniform partitioning. The nonuniform partitioning is done using different values for the parameter k in Formulas 4.8.

Table 5.4 compares the results obtained from solving the CC and NCC problems, with $k = 1.5, 1.7$. For each model, the solving time is listed in seconds or as TL if the time limit is reached. The objective value of the best integer solution found within the time limit is listed under BI, and F means the solver failed to find an integer solution. Initially, 10 breakpoints were used by the models, but CPLEX failed to find a solution for most of the problems, so they were regenerated using 7 points. The third column measures the difference, if applicable, between the best integer value, $\overline{f^*}$ and the best value of the original instance, f^*, which can be found in Table 5.1. Smaller $|f^* - \overline{f^*}|$ value indicates that the approximation is acceptable. Note that for problems that reached time limit, the best integer value could keep improving if there is no time limit.

Given the fact that most of the original instances were solved by SCIP to global optimality within less than an hour (Table 5.1), it can be confirmed that complete

Table 5.3: The Sizes of the Instances Produced by CC and LOG Models with 10 Breakpoints.

p#	CC			LOG		
	BV	CV	Cons	BV	CV	Cons
1	6966	4318	4526	344	25818	871
2	6966	4318	4526	344	25818	871
3	9234	5720	5991	456	34220	1152
4	11826	7322	7678	584	43822	1473
5	1134	710	744	56	4210	149
6	1134	710	744	56	4210	149
7	2106	1312	1376	104	7812	271
8	2106	1312	1376	104	7812	271
9	3402	2114	2218	168	12614	433
10	3402	2114	2218	168	12614	433
11	5022	3116	3270	248	18616	635
12	5022	3166	3270	248	18616	635
13	6966	4318	4532	344	25818	877
14	14742	9124	9535	728	54624	1800
15	3402	2114	2208	168	12614	423
16	3402	2114	2208	168	12614	423
17	5022	3118	3255	248	18616	620
18	6966	4318	4512	344	25818	857
19	9234	5720	5977	456	34220	1132
20	11826	7322	7652	584	43822	1447

Table 5.4: Results of Solving MILP Problems Produced by CC and NCC Models Using 7 Breakpoints.

p#	CC			NCC ($k = 1.5$)			NCC ($k = 1.7$)		
	time	BI	$\lvert f^* - \overline{f^*}\rvert$	time	BI	$\lvert f^* - \overline{f^*}\rvert$	time	BI	$\lvert f^* - \overline{f^*}\rvert$
1	TL	F	NA	TL	1.184	0.9277	TL	2.47	0.3583
2	TL	F	NA	TL	3.324	1.3504	TL	F	NA
3	TL	F	NA	TL	2.0317	0.6311	TL	2.6628	0
4	TL	F	NA	TL	F	NA	TL	F	NA
5	18	0.5724	0.0714	29	0.6009	0.0429	81	0.5847	0.0591
6	11	1.366	0.0099	8	1.3831	0.0072	5	1.3544	0.0215
7	1	0.8084	0.05	1	0.8584	0	1	0.8584	0
8	TL	0.8084	0.05	133	0.8269	0.0315	110	0.8052	0.0532
9	TL	1.5373	0.4643	TL	0.7367	0.3363	TL	0.5087	0.5643
10	1200	1.473	0.0641	TL	1.5371	0	TL	1.588	0.0509
11	TL	1.221	0.0666	TL	1.2876	0	TL	1.2876	0
12	856	1.118	0.1696	TL	1.2876	0	TL	1.2876	0
13	TL	1.6689	0.2974	TL	1.9663	0	TL	1.817	0.1493
14	TL	F	NA	TL	F	NA	TL	F	NA
15	TL	5.2619	0.1457	TL	5.1162	0	TL	6.479	1.3628
16	1400	2.838	2.2782	527	3.172	1.9442	TL	3.4476	1.6686
17	3400	7.5885	0.1994	TL	7.2298	0.558	TL	F	NA
18	TL	F	NA	TL	F	NA	TL	F	NA
19	TL	F	NA	TL	15.431	0.9497	TL	17.611	3.1297
20	TL	F	NA	TL	F	NA	TL	F	NA

transformation of MINLP problems into MILP ones will not make the problems easier, especially when approximating many functions. Nonetheless, this is not the goal of the experiments since the main purpose is to compare partitioning methods for PLA models. The aspects of the comparison between uniform and nonuniform partitioning will be the accuracy of the approximation and the time needed to solve the approximation.

The data listed under the time columns indicate that all models are close in terms of the number of problems that were solved within the time limit. However, for problems that reached the time limit, the model with uniform partitioning failed to find an integer solution in eight problems, compared to four problems for the NCC model with $k = 1.5$. This implies that problems generated by CC models with nonuniform partitioning usually find feasible solutions faster than with uniform partitioning. The same outcome resulted when other nonuniform cases were tested for $k = 1.4, 1.6, 1.8$. Even when both uniform and nonuniform cases resulted in a problem that produced an integer solution within the time limit, the gap in the nonuniform case is smaller most of the times. Therefore, it can be concluded that problems produced by NCC models are usually solved faster than the ones produced by CC models.

The quality of the approximation is better tested without the time limit, where the solver runs until the global solution is found, and then the solutions of the original and approximated problems are compared. In spite of that, the table provides enough data to compare the quality of CC and NCC approximations. It can be observed that the integer solution is the same as the solution of the original instance in many problem for both cases of nonuniform partitioning, while the uniform partitioning never produced an identical integer solution to the original one. Also for the cases

that the integer solution is not the same as the original one, the difference between the two solutions is mostly less in the NCC cases.

The computational results for solving problems produced by the models LOG and NLOG are summarized in Table 5.5. The test setting and comparison aspects for these models are the same as the CC and NCC models, except for the number of breakpoints where 11 is used for this test. The outcomes also turned out to be similar: the nonuniform partitioning improved the models in both speed and accuracy. Both NLOG models resulted in more problems that were solved within time limit, compared to LOG models; and when a problem is solved within time limit for all models, almost always the NLOG problem needs less solving time. Also the difference $|f^* - \overline{f^*}|$ is mostly smaller in the NLOG cases. It should be mentioned that in nonuniform partitioning, having a local solution as one of the breakpoints helped improving the model's quality.

Comparing between the data in Tables 5.5 and 5.4 makes it clear that the logarithmic models are significantly better than the non logarithmic ones. This confirms the fact that for a PLA model, increasing the number of binary variables has more negative impact than increasing the continuous variables. Even though the logarithmic models used 4 points more (11 compared to 7), which considerably affects the size, the solving times turned out to be much less than the times for the non logarithmic case. Therefore, using logarithmic models allows introducing more breakpoints and, consequently, producing more accurate approximation. For example, the average difference between optimal value of the original instance and approximation, $|f^* - \overline{f^*}|$, per LOG problem is 0.328, while it is 0.667 for CC problems.

The computational experiments in this section have shown that full PLA of MINLP problems will not make the new problems any better, yet they prove that the improvement of PLA models is promising. In addition to the major development of

Table 5.5: Results of Solving MILP Problems Produced with LOG and NLOG Models Using 11 Breakpoints.

p#	LOG			NLOG ($k = 1.5$)			NLOG ($k = 1.7$)		
	time	BI	$\lvert f^* - \overline{f^*}\rvert$	time	BI	$\lvert f^* - \overline{f^*}\rvert$	time	BI	$\lvert f^* - \overline{f^*}\rvert$
1	TL	F	NA	TL	F	NA	TL	F	NA
2	TL	F	NA	TL	1.9488	0.0248	TL	F	NA
3	TL	F	NA	TL	2.6628	0	TL	2.6628	0
4	TL	F	NA	TL	F	NA	TL	F	NA
5	8	0.5435	0.1003	6	0.6009	0.0429	73	0.5847	0.0591
6	10	1.368	0.0079	2	1.3831	0.0072	157	1.3544	0.0215
7	3	0.8324	0.026	1	0.8584	0	47	0.8584	0
8	37	0.8348	0.0236	16	0.8269	0.0315	133	0.8052	0.0532
9	TL	2.944	1.871	TL	1.036	0.037	TL	0.378	0.695
10	1764	1.51	0.0271	194	1.5371	0	915	1.5371	0
11	TL	1.651	0.3634	1042	1.2876	0	821	1.2876	0
12	346	1.038	0.2496	231	1.2876	0	478	1.2876	0
13	TL	2.097	0.1307	TL	1.9663	0	TL	1.9663	0
14	TL	F	NA	TL	F	NA	TL	F	NA
15	2980	5.038	0.0782	TL	5.1162	0	161	4.6298	0.4864
16	TL	4.201	0.9152	191	3.172	1.9442	1625	3.4476	1.6686
17	TL	7.642	0.1459	450	7.2298	0.5581	2516	7.572	0.2159
18	TL	F	NA	TL	6.411	0.7421	TL	F	NA
19	TL	F	NA	935	7.182	7.2993	TL	2.288	12.1933
20	TL	F	NA	TL	7.217	6.1333	TL	F	NA

the PLA model caused by logarithmic formulation, different components of the model can also be improved. For example, it was demonstrated that nonuniform partitioning based on local solutions can add more improvement to the models. Finally, the uses of PLA are not limited to the full PLA; it can be useful if applied to only parts of an MINLP problem, as will be shown in the next section.

5.1.2 Partial PLA

PLA was applied partially to different QCQP and MIQCQP using Algorithm 1. If the number of approximated constraints is large, then the modified problem gets more complicated and the original problem is better off without this approximation. However, in this section, the algorithm is applied with one iteration to many instances belonging to different groups from the MINLPLib library. Interestingly, this approach worked on some instances, and produced problems that needed shorter solving times. Table 5.6 summarizes the statistics of some of these instances, in addition to computational results of solving them using SCIP with 2 hours time limit.

Table 5.6: Statistics of Different Test Instances, and the Solving Results by SCIP.

instance	seq#	#v	#cons (q)	obj val	time	gap %
kall_circlespolygons_c1p12	1	43	48 (21)	0.3396	TL	∞
kall_circlespolygons_c1p13	2	43	48 (21)	0.3396	TL	∞
kall_circlesrectangles_c1r12	3	49	41 (23)	0.3396	TL	∞
kall_diffcircles_5a	4	14	24 (11)	5.1162	63	0
kall_diffcircles_6	5	16	31 (16)	7.7879	102	0
pooling_foulds3stp	6	832	1089 (1024)	-8	5730	0
pooling_foulds4stp	7	832	1089 (1024)	-8	6235	0

As can be seen in the table, SCIP failed to close the gap when solving instances 1, 2, and 3; but it succeeded for the other problem. The goal of testing these instances is to show that approximating one constraint per instance aids the solver to handle the instance better. For each modified instance Table 5.7 presents the solving time, the optimal objective value $\overline{f^*}$, and the difference between the optimal values of the original and modified instance.

Table 5.7: Computational Results of Problems Produced by Partial PLA Using 10 Breakpoints.

p#	CC			LOG		
	time	$\overline{f^*}$	$\lvert f^* - \overline{f^*}\rvert$	time	$\overline{f^*}$	$\lvert f^* - \overline{f^*}\rvert$
1	92	0.2949	0.0447	241	0.2848	0.0548
2	258	0.2914	0.0482	358	0.3227	0.0169
3	TL	0.3396	0	TL	0.3396	0
4	38	4.96	0.1562	29	4.96	0.1562
5	68	7.7879	0	47	7.7879	0
6	215	-8	0	2631	-8	0
7	3143	-8	0	622	-8	0

The results table shows that for the first three instances, SCIP was able to solve them and close the gap, except for the third instance, the gap was closed shortly after 2 hours (when time limit was disabled) for the cc case, and the gap was $\infty\%$ in the LOG case. The solving time of the modified instance is mostly shorter than the original problem.

From instances with $|f^* - \overline{f^*}| > 0$, it can be implied that even if it is only one out of many constraints that was approximated, there will probably be an approximation error. These instances were approximated again with 15, 20, and 25 breakpoints,

and that led to better optimal values but longer solving time. For example, when instance 1 was generated by CC with 15 points, SCIP needed 250 seconds to solve it and the objective value was 0.311; and with 25 points, it needed 440 seconds, yielding an objective value of 0.334.

An interesting finding is that the solving times of CC problems are shorter than those of LOG problems. This agrees with what was mentioned in the comparison section in Chapter 4, which is that the models have similar performance when approximating problems with few variables using few breakpoints.

The experimental results suggest that partial PLA can produce less complicated problems with small or zero approximation error. Also, it was shown that LOG models have no advantage here in contrary to the case in the previous section. In the following section, similar computational experiments are performed on MIQCP problems.

5.2 Mixed Integer Variables

The PLA models, that were used in the previous section, were applied to problems where the variables involved in the quadratic terms can be integer. When introducing the breakpoints to an integer variable's interval, every integer value in the interval will be a breakpoint, so the number of breakpoints depends on the length of the interval. If this results in a large number of breakpoints, some integer values can be skipped so the number stays reasonable. Also there is no need to have a convex combination of two breakpoints since the variable cannot take noninteger values. For all noninteger variables involved in the quadratic functions, the PLA procedures are still the same.

The objective of the computational tests is the same as in the case of continuous variables: to test the uniform against the nonuniform partitioning, and to assess the performance of the partial PLA models. A summary of the test instances' statistics

is given in Table 5.8, where they were solved by SCIP with a time limit of two hours. Instances 1, 2, and 3 were taken from the QPLIB library and the remainder are from MINLPLib instances.

Table 5.8: Statistics of MIQCP Test Instances, and the Solving Results by SCIP.

instance	seq#	#v	#cons (q)	obj val	time	gap %
3562	1	63	42 (7)	15	TL	305
3780	2	168	72 (12)	90.6	TL	1138.3
3816	3	187	387 (24)	7.3936	TL	27.37
blend029	4	102	213 (12)	13.359	8	0
blend146	5	222	624 (24)	45.297	TL	1.87
ex1236	6	92	55 (4)	19.6	1	0
gabriel02	7	261	597 (96)	39.6	TL	22.3
sep1	8	29	31 (6)	-510.81	1	0
st_e31	9	112	135 (5)	-2	4	0
tln4	10	24	24 (4)	8.3	6	0

Based on the solving times in Table 5.8, it can be concluded that half of the tested instances are challenging and the other half are not. It should be mentioned that in instances 1, 2, and 10, only integer variables are involved in the quadratic functions, so no nonuniform partitioning is done for these instances (a breakpoint is introduced at every integer value in the interval). Tables 5.9 and 5.10 show the results of solving the problems produced by the convex combination and the logarithmic models with 11 breakpoints. The computations were done by using CPLEX with 2 hours time limit.

The results in the tables show that CPLEX failed to find an integer solution to 3 NLOG problems and 4 problems produced by the other models. For problems for

Table 5.9: Results of Solving MILP Problems Produced by CC and NCC Models Using 11 Breakpoints.

p#	CC			NCC ($k = 1.5$)			NCC ($k = 1.7$)		
	time	BI	$\lvert f^* - \overline{f^*}\rvert$	time	BI	$\lvert f^* - \overline{f^*}\rvert$	time	BI	$\lvert f^* - \overline{f^*}\rvert$
1	TL	19.6	4.6	similar					
2	TL	F	NA	similar					
3	TL	F	NA	TL	F	NA	TL	F	NA
4	TL	12.73	0.629	TL	12.73	0.629	TL	13.359	0
5	TL	F	NA	TL	F	NA	TL	F	NA
6	360	19.6	0	68	19.6	0	237	19.6	0
7	TL	F	NA	TL	F	NA	TL	F	NA
8	109	-510.29	0.25	88	-510.08	0.73	21	-510.08	0.73
9	331	-2.019	0.019	29	-2.087	0.087	16	-2.188	0.188
10	TL	8.3	0	similar					

which CPLEX found integer solutions, it can be observed that the problems with nonuniform partitioning are solved faster than the uniformly partitioned problems, for both logarithmic and nonlogarithmic models. The data under $|f^* - \overline{f^*}|$ columns show that no clear advantage can be confirmed for any of the models in the accuracy of the approximation.

Partial PLA was applied to the same set of instances and the most violated constraint of each challenging instance was approximated using CC and NLOG models. As Table 5.8 shows, instances 1 and 2 have a relative gap of 305% and 1138%, respectively, after two hours of solving time. After PLA, SCIP reached the time limit and resulted in gaps of 53% and 1862% for the CC problems, and 200% and 774%

Table 5.10: Results of Solving MILP Problems Produced with LOG and NLOG Models Using 11 Breakpoints.

p#	LOG			NLOG ($k = 1.5$)			NLOG ($k = 1.7$)		
	time	BI	$\lvert f^* - \overline{f^*}\rvert$	time	BI	$\lvert f^* - \overline{f^*}\rvert$	time	BI	$\lvert f^* - \overline{f^*}\rvert$
1	TL	18.4	3.4	similar					
2	TL	F	NA	similar					
3	TL	F	NA	TL	6.6	0.7936	TL	6.2	1.1936
4	5088	13.359	0	756	13.359	0	970	13.359	0
5	TL	F	NA	TL	F	NA	TL	F	NA
6	146	19.6	0	29	19.6	0	41	19.6	0
7	TL	F	NA	TL	F	NA	TL	F	NA
8	103	-509.72	1.09	34	-510.08	0.73	43	-510.08	0.73
9	198	-2.01	0.01	89	-2.087	0.087	103	-2.188	0.188
10	354	8.3	0	similar					

for the LOG problems. The rest of instances resulted in better performances by the solver without PLA.

From the results of the computational experiments performed for both continuous (Section 5.1) and mixed integer problems, it can be concluded that PLA models can be improved by using nonuniform instead of uniform partitioning. Also, the tests have shown that some challenging QCP and MIQCP problems can be less challenging with only a small change to them by applying partial PLA.

5.3 MINLP 2d Examples

In this section, PLA will be applied to five MINLP problems with functions of two variables. The functions in these problems are characterized by their many local

minima and maxima. The problems are not necessarily transformed to MILP problems, but the PLA will be applied to functions to either get simpler problems, or to give some solvers the ability to solve problems having unsupported functions. For example, BARON and SCIP do not support trigonometric functions, so PLA will convert them to supported forms. The problems are given by the following examples:

Example 5.1

$$max\ |\frac{(\cos(x)^4\cos(y)^4 - 2\cos(x)^2\cos(y)^2)}{\sqrt{(}x^2+2y^2)}| \tag{5.1a}$$

$$subject\ to\ -xy + 0.75 \leq 0 \tag{5.1b}$$

$$x + y - 15 \leq 0. \tag{5.1c}$$

Example 5.2

$$\begin{aligned} min\ &(\cos((x-0.1)y))^2 - x\sin(3x+y) \\ subject\ to\ &x^2+y^2 \leq (2cos(\frac{2y}{x})) - 0.5\cos(2\frac{2y}{x}) \\ &\quad - 0.25\cos(3\frac{2y}{x}) - 0.125\cos(4\frac{2y}{x})^2 \\ &\quad + (2\sin(\frac{2y}{x}))^2. \end{aligned} \tag{5.2a, 5.2b}$$

Example 5.3

$$min\ \sin(y)e^{(1-\cos(x))^2} + \cos(x)e^{(1-\sin(y))^2} + (x-y)^2 \tag{5.3a}$$

$$subject\ to\ (x+5)^2 + (y+5)^2 \leq 25. \tag{5.3b}$$

Example 5.4

$$min\ (4 - 2.1x^2 + x^{\frac{4}{3}})x^2 + xy + (-4+4y^2)y^2 \tag{5.4a}$$

$$subject\ to\ -\sin(4\pi x) + (\sin(2\pi y))^2 \leq 0. \tag{5.4b}$$

Example 5.5

$$max\ \frac{(sin(2\pi x))^3 sin(2\pi y)}{x^3(x+y)} \tag{5.5a}$$

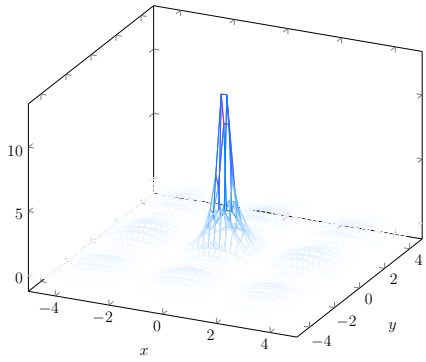
$$subject\ to\ x^2 - y + 1 \leq 0 \tag{5.5b}$$

$$1 - x + (y-4)^2 \leq 0. \tag{5.5c}$$

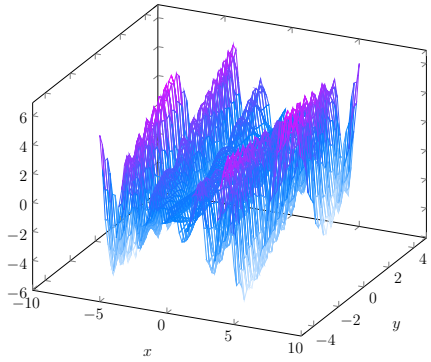
The objective functions of the optimization problems, mentioned in the examples above, are illustrated in Figure 5.1. Since the problems have only two variables, it is expected that they will be handled easily by any global solver. Note that even though the objective function in Example 4 looks simple, the constraint adds some complexity to the problem.

The objective of the computational tests performed on these problems is to show how PLA can be useful, even to some global solvers such as BARON and SCIP. Currently, these two solvers cannot solve the problems because all problems contain trigonometric functions either in the objective function or in the constraints. Trigonometric functions are involved in many real world problems, and it could be disappointing that two of the most powerful solvers do not handle them. Therefore, an attempt was made to apply PLA models to these problems so they could be handled by the solvers. The CC model was applied to the trigonometric functions in the problems, and the other nonlinear functions remain unchanged. Consequently, trigonometric functions were replaced by piecewise linear ones and both BARON and SCIP were able to handle the problems. Also the PLA was applied to all nonlinear functions to transform the problems to MILP so CPLEX and other linear solvers can handle them.

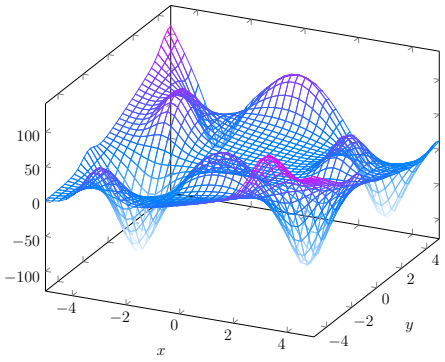
Table 5.11 shows the global optimal solutions of the original problems, where they were solved using Couenne, and their approximations that were solved by BARON and SCIP. The variable bounds for all problems were set such that the interval length



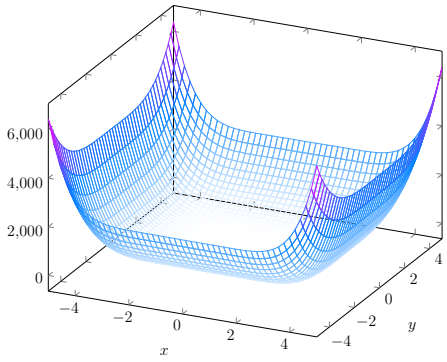
(a) The Objective Function of Problem 5.1



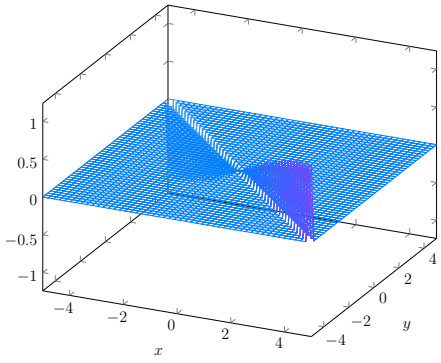
(b) The Objective Function of Problem 5.2



(c) The Objective Function of Problem 5.3



(d) The Objective Function of Problem 5.4



(e) The Objective Function of Problem 5.5

Figure 5.1: Different Nonlinear Objective Functions with Two Variables.

is 2 and the global solution is in the interval. The PLA was done using the CC model with 10 and 30 breakpoints, and it can be seen from the table that both models resulted in good approximations, with an advantage to the 30 breakpoints model. All variations of the problems were solved by the NEOS server's solvers with their default settings.

Table 5.11: The Examples' Global Solutions and Optimal Values Before (Couenne) and After PLA.

		Couenne	PLA (CC)	
			10	30
Ex.1	(x, y)	(2.722, 0.276)	(2.719, 0.276)	(2.71, 0.277)
	f^*	0.344815	0.34236	0.3445
Ex.2	(x, y)	(-2.074. -1.0714)	(-2.05, -1.11)	(-2.055. -1.103)
	f^*	-2.22789	-2.1845	-2.2275
Ex.3	(x, y)	(-3.123, -1.589)	(-3.11, -1.67)	(-3.103, -1.62)
	f^*	-106.788	-106.284	-106.68
Ex.4	(x, y)	(0.097, -0.71)	(0.089, -0.71)	(0.097, -0.71)
	f^*	-1.03136	-1.0316	-1.03134
Ex.5	(x, y)	(1.228. 4.245)	(1.336,4.33)	(1.245, 4.241)
	f^*	0.095825	0.04	0.09414

Each problem was solved many times and the average solving times were recorded in Table 5.12. Although the problems are simple and it took no more than a few seconds to solve the variations of each problem, some interesting findings can be observed. For example, the improvement of the approximation gained by adding

more breakpoints might not be worth the increase in the solving time. The solving time for BARON and SCIP using 10 points is comparable to Couenne's, but increasing the points to 30 gave Couenne the advantage, and this advantage would be greater if the variables' interval length is greater than 2. Another interesting observation is that the performance of CPLEX in the MILP problems with 30 points is even better than the performance of Couenne in the original problems. This might indicate that using PLA on small MINLP problems might produce MILP problems that are easier to be dealt with, although it is hard to conclude this by only testing these trivial problems.

Table 5.12: The Time Needed by Solvers to Solve the Original Problems and their Approximations(in seconds).

solver	#bp	Ex.1	Ex.2	Ex.3	Ex.4	Ex.5
Couenne		5.5	10.5	0.2	0.08	0.12
BARON	10	0.4	1	0.1	3.1	0.27
	30	10	18	3.2	37.2	2.4
SCIP	10	0.5	7	0.08	2.9	0.28
	30	7.2	6.3	3.6	9	4
CPLEX	10	0.02	0.1	0.01	0.01	0.11
	30	0.2	1.2	0.11	0.02	0.9

Chapter 6

CONCLUSIONS

This dissertation has discussed optimization in general and global MINLP in detail. It was shown that LP, MILP, and NLP methods are proved to be efficient, so they were adapted to be the main parts of the MINLP methods. The most important solvers that employed these methods were inspected and computational results that compare their performance were provided.

Piecewise linear approximation is a method that transforms a nonlinear function to a piecewise linear one, and it is applied to MINLP problems to allow them to be handled by robust MILP solvers. The PLA background was given and the models were described for one and two dimensional functions.

While it is desired to get good approximation, it should not come at the expense of creating a large problem. Therefore, this dissertation suggests some modifications to the PLA models to produce better approximation without affecting the size of the problem. Also the dissertation suggests another PLA approach that improves the performance of some MINLP solvers on some problems.

It was demonstrated through computational experiments that the approaches introduced in Chapter 4 are promising. The PLA models that were used for the experiments are coded in a Python interface for the solver SCIP. Four codes were written and they can transform any MIQCQP problem into an MILP one through SCIP. Also a PLA model was applied to MINLP problems with trigonometric functions which

allows SCIP and BARON to solve the approximated problems, while they do not handle the original ones because of the trigonometric functions.

6.1 Future Work

PLA was proved to be a promising method to tackle MINLP problems, and it can be improved in many areas. In Section 4.4.1, a nonuniform partitioning is introduced to improve the quality of the approximation. This can be improved by studying the approximated function and choosing a relevant partitioning formula. Also the partial PLA, Section 4.4.2, can be improved by modifying the constraints selection rule. In this dissertation, the selection rule is to select the most violated constraint and approximate it. Partial PLA using this rule improved SCIP in 10 to 20% of the tested examples, so other selection ideas may increase the rate.

The written codes implement the convex combination and logarithmic disaggregated convex combination models, for one and two dimensions, and one triangulation (Figure 4.4a). The codes can be modified to allow to pick from more than one triangulation, and more codes can be written to implement the other models. Also they can be extended to be applicable to the NLP and MINLP problems through SCIP.

Achterberg, T., "SCIP-a framework to integrate constraint and mixed integer programming", (2004).

Achterberg, T., "SCIP: solving constraint integer programs", Mathematical Programming Computation **1**, 1, 1–41 (2009).

Achterberg, T., T. Koch and A. Martin, "Branching rules revisited", Operations Research Letters **33**, 1, 42–54 (2005).

Adjiman, C. S., I. P. Androulakis and C. A. Floudas, "Global optimization of mixed-integer nonlinear problems", AIChE Journal **46**, 9, 1769–1797 (2000).

Adjiman, C. S., S. Dallwig, C. A. Floudas and A. Neumaier, "A global optimization method, αBB, for general twice-differentiable constrained NLPs-I. Theoretical advances", Computers & Chemical Engineering **22**, 9, 1137–1158 (1998).

Al-Khayyal, F. A. and J. E. Falk, "Jointly constrained biconvex programming", Mathematics of Operations Research **8**, 2, 273–286 (1983).

Androulakis, I. P., C. D. Maranas and C. A. Floudas, "αBB: A global optimization method for general constrained nonconvex problems", Journal of Global Optimization **7**, 4, 337–363 (1995).

Applegate, D. and R. Bixby, *Finding cuts in the TSP (A preliminary report)*, vol. 95 (Citeseer, 1995).

Applegate, D., R. Bixby, W. Cook and V. Chvátal, "On the solution of traveling salesman problems", (1998).

Balakrishnan, A. and S. C. Graves, "A composite algorithm for a concave-cost network flow problem", Networks **19**, 2, 175–202 (1989).

Beale, E. and J. J. Forrest, "Global optimization using special ordered sets", Mathematical Programming **10**, 1, 52–69 (1976).

Beale, E. and J. Tomlin, "Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables", OR **69**, 447-454, 99 (1970).

Belotti, P., C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke and A. Mahajan, "Mixed-integer nonlinear optimization", Acta Numerica **22**, 1–131 (2013).

Belotti, P., J. Lee, L. Liberti, F. Margot and A. Wächter, "Branching and bounds tightening techniques for non-convex MINLP", Optimization Methods & Software **24**, 4-5, 597–634 (2009).

Bénichou, M., J.-M. Gauthier, P. Girodet, G. Hentges, G. Ribière and O. Vincent, "Experiments in mixed-integer linear programming", Mathematical Programming **1**, 1, 76–94 (1971).

Berthold, T., G. Gamrath, A. M. Gleixner, S. Heinz, T. Koch and Y. Shinano, "Solving mixed integer linear and nonlinear problems using the SCIP optimization suite", ZIB-Report 12-27 pp. 1–23 (2012).

Bonami, P., L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya *et al.*, "An algorithmic framework for convex mixed integer nonlinear programs", Discrete Optimization **5**, 2, 186–204 (2008).

Bussieck, M. R. and S. Vigerske, "MINLP solver software", (2010).

Croxton, K. L., B. Gendron and T. L. Magnanti, "A comparison of mixed-integer programming models for nonconvex piecewise linear cost minimization problems", Management Science **49**, 9, 1268–1273 (2003).

Dahl, G. and B. Realfsen, "Curve approximation and constrained shortest path problems", Preprint (Universitetet i Oslo, Institutt for informatikk) http://urn. nb. no/URN: NBN: no-35455 (1996).

D'Ambrosio, C., A. Lodi and S. Martello, "Piecewise linear approximation of functions of two variables in MILP models", Operations Research Letters **38**, 1, 39–46 (2010).

Dantzig, G. B., "On the significance of solving linear programming problems with some integer variables", Econometrica, Journal of the Econometric Society pp. 30–44 (1960).

Duran, M. A. and I. E. Grossmann, "An outer-approximation algorithm for a class of mixed-integer nonlinear programs", Mathematical programming **36**, 3, 307–339 (1986).

Falk, J. E. and R. M. Soland, "An algorithm for separable nonconvex programming problems", Management science **15**, 9, 550–569 (1969).

Fletcher, R. and S. Leyffer, "Solving mixed integer nonlinear programs by outer approximation", Mathematical programming **66**, 1-3, 327–349 (1994).

Forrest, J. J. and J. A. Tomlin, "Branch and bound, integer, and non-integer programming", Annals of Operations Research **149**, 1, 81–87 (2007).

Geißler, B., A. Martin, A. Morsi and L. Schewe, "Using Piecewise Linear Functions for Solving MINLPs", in "Mixed integer nonlinear programming", pp. 287–314 (Springer, 2012).

Gleixner, A. M. and S. Weltge, "Learning and propagating Lagrangian variable bounds for mixed-integer nonlinear programming", in "International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems", pp. 355–361 (Springer, 2013).

Gomory, R. E., "Outline of an algorithm for integer solutions to linear programs", Bull. Amer. Math. Soc. **64**, 5, 275–278, URL `https://projecteuclid.org:443/euclid.bams/1183522679` (1958).

Horst, R., "Deterministic methods in constrained global optimization: Some recent advances and new fields of application", Naval Research Logistics (NRL) **37**, 4, 433–471 (1990).

Jeroslow, R. G. and J. K. Lowe, "Modelling with integer variables", in "Mathematical Programming at Oberwolfach II", pp. 167–184 (Springer, 1984).

Kelley, J. E., Jr, "The cutting-plane method for solving convex programs", Journal of the society for Industrial and Applied Mathematics **8**, 4, 703–712 (1960).

Kesavan, P., R. J. Allgor, E. P. Gatzke and P. I. Barton, "Outer approximation algorithms for separable nonconvex mixed-integer nonlinear programs", Mathematical Programming **100**, 3, 517–535 (2004).

Kesavan, P. and P. I. Barton, "Generalized branch-and-cut framework for mixed-integer nonlinear optimization problems", Computers & Chemical Engineering **24**, 2-7, 1361–1366 (2000).

Kılınç, M. R. and N. V. Sahinidis, "Exploiting integrality in the global optimization of mixed-integer nonlinear programming problems with BARON", Optimization Methods and Software pp. 1–23 (2017).

Land, A. H. and A. G. Doig, "An automatic method of solving discrete programming problems", Econometrica: Journal of the Econometric Society pp. 497–520 (1960).

Lin, Y. and L. Schrage, "The global solver in the LINDO API", Optimization Methods & Software **24**, 4-5, 657–668 (2009).

Maranas, C. D. and C. A. Floudas, "Global minimum potential energy conformations of small molecules", Journal of Global Optimization **4**, 2, 135–170 (1994).

Markowitz, H. M. and A. S. Manne, "On the solution of discrete programming problems", Econometrica: journal of the Econometric Society pp. 84–110 (1957).

McCormick, G. P., "Converting general nonlinear programming problems to separable nonlinear programming problems", Tech. rep., GEORGE WASHINGTON UNIV WASHINGTON DC PROGRAM IN LOGISTICS (1972).

McCormick, G. P., "Computability of global solutions to factorable nonconvex programs: Part I-Convex underestimating problems", Mathematical programming **10**, 1, 147–175 (1976).

Meyer, R. R., "Mixed integer minimization models for piecewise-linear functions of a single variable", Discrete Mathematics **16**, 2, 163–171 (1976).

Misener, R. and C. A. Floudas, "GloMIQO: Global mixed-integer quadratic optimizer", Journal of Global Optimization **57**, 1, 3–50 (2013).

Misener, R. and C. A. Floudas, "ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations", Journal of Global Optimization **59**, 2-3, 503–526 (2014).

Mittelmann, H., "Mixed Integer Nonlinear Programming Benchmark", URL `http://plato.asu.edu/ftp/minlp.html` (2020).

Morrison, D. R., S. H. Jacobson, J. J. Sauppe and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning", Discrete Optimization **19**, 79–102 (2016).

Nagarajan, H., M. Lu, S. Wang, R. Bent and K. Sundar, "An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs", Journal of Global Optimization **74**, 4, 639–675 (2019).

Padberg, M., "Approximating separable nonlinear functions via mixed zero-one programs", Operations Research Letters **27**, 1, 1–5 (2000).

Padberg, M. and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems", SIAM review **33**, 1, 60–100 (1991).

Rebennack, S. and J. Kallrath, "Continuous piecewise linear delta-approximations for univariate functions: computing minimal breakpoint systems", Journal of Optimization Theory and Applications **167**, 2, 617–643 (2015).

Ryoo, H. S. and N. V. Sahinidis, "Global optimization of nonconvex NLPs and MINLPs with applications in process design", Computers & Chemical Engineering **19**, 5, 551–566 (1995).

Ryoo, H. S. and N. V. Sahinidis, "A branch-and-reduce approach to global optimization", Journal of Global Optimization **8**, 2, 107–138 (1996).

Sahinidis, N. V., "BARON: A general purpose global optimization software package", Journal of global optimization **8**, 2, 201–205 (1996).

Sherali, H. D. and A. Alameddine, "A new reformulation-linearization technique for bilinear programming problems", Journal of Global optimization **2**, 4, 379–410 (1992).

Smith, E. M. and C. C. Pantelides, "Global optimisation of nonconvex MINLPs", Computers & Chemical Engineering **21**, S791–S796 (1997).

Smith, E. M. and C. C. Pantelides, "A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs", Computers & Chemical Engineering **23**, 4-5, 457–478 (1999).

Sridhar, S., J. Linderoth and J. Luedtke, "Locally ideal formulations for piecewise linear functions with indicator variables", Operations Research Letters **41**, 6, 627–632 (2013).

Tawarmalani, M. and N. V. Sahinidis, *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications*, vol. 65 (Springer Science & Business Media, 2002).

Tawarmalani, M. and N. V. Sahinidis, "Global optimization of mixed-integer nonlinear programs: A theoretical and computational study", Mathematical programming **99**, 3, 563–591 (2004).

Tawarmalani, M. and N. V. Sahinidis, "A polyhedral branch-and-cut approach to global optimization", Mathematical Programming **103**, 2, 225–249 (2005).

Trespalacios, F. and I. E. Grossmann, "Review of mixed-integer nonlinear and generalized disjunctive programming methods", Chemie Ingenieur Technik **86**, 7, 991–1012 (2014).

Vasudeva, V., *Global optimization with piecewise linear approximation*, Ph.D. thesis, The University of Texas at Austin (2015).

Vielma, J. P., S. Ahmed and G. Nemhauser, "Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions", Operations research **58**, 2, 303–315 (2010).

Vielma, J. P. and G. L. Nemhauser, "Modeling disjunctive constraints with a logarithmic number of binary variables and constraints", Mathematical Programming **128**, 1-2, 49–72 (2011).

Vigerske, S., *Decomposition in multistage stochastic programming and a constraint integer programming approach to mixed-integer nonlinear programming*, Ph.D. thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II (2013).

Vigerske, S. and A. Gleixner, "SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework", Optimization Methods and Software **33**, 3, 563–593 (2018).

Viswanathan, J. and I. E. Grossmann, "A combined penalty function and outer-approximation method for MINLP optimization", Computers & Chemical Engineering **14**, 7, 769–782 (1990).

Westerlund, T. and F. Pettersson, "An extended cutting plane method for solving convex MINLP problems", Computers & Chemical Engineering **19**, 131–136 (1995).