The Dissertation Committee for Vivek Vasudeva Certifies that this is the approved version of the following dissertation:

Global Optimization with Piecewise Linear Approximation

Committee:

Anantaram Balakrishnan, Supervisor

Leon Lasdon, Co-Supervisor

Jonathan F. Bard

Guoming Lai

Efstathios Tompaidis

Global Optimization with Piecewise Linear Approximation

by

Vivek Vasudeva, B. Tech., M.S.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2015

Dedication

This dissertation is dedicated to my parents and my sisters Madhvi and Priyanka whose constant support helped me make it to and through graduate school.

Acknowledgements

I would like to express my sincere gratitude to my advisers Prof. Anant Balakrishnan and Prof. Leon Lasdon for their constant support and encouragement at every step in the program. I thank my friends Alvin, Kathy, Qi, Qian, Vishwakant, Wen, Yanzhen and Zhuoxin in the IROM department for providing reassurance that was often required to handle the stresses of student life. I also thank my friends Dhaval, Rajani and Jyoti who have influenced my life ever since I started my graduate studies. A special thanks to Tarun for reviewing my documents and for providing intellectually stimulating conversations. Finally, a note of appreciation for Cooper, Hyun, Laura, Lucy, Maria, Patricia, Samantha and Vince—my friends at the local coffee shop—for cheering me with their coffee in the mornings and the afternoons.

Global Optimization with Piecewise Linear Approximation

Vivek Vasudeva, PhD

The University of Texas at Austin, 2015

Supervisors: Anant Balakrishnan and Leon Lasdon

Global optimization deals with the development of solution methodologies for nonlinear nonconvex optimization problems. These problems, which could arise in diverse situations ranging from optimizing hydro-power generation schedules to estimating coefficients of non-linear regression models, are difficult for traditional nonlinear solvers that iteratively search the neighborhood around a starting point. The Piecewise Linear Approximation (PLA) method that we study in this dissertation seeks to generate 'good' starting points, hopefully ones that lie in the basin of attraction of the globally optimal solution. In this approach, we approximate the non-linear functions in the optimization problem by piecewise linear functions defined over the vertices of a grid that partitions the domain of each nonlinear function into cells. Based on this approximation, we convert the original nonlinear program into a mixed integer program (MIP) and use the solution to this MIP as a starting point for a local nonlinear solver. In this dissertation, we validate the effectiveness of the PLA approach as a global optimization approach by applying it to a diverse set of continuous and discrete nonlinear optimization problems. Further, we develop various modeling and algorithmic strategies for enhancing the basic approach. Our computational results demonstrate that the PLA approach works well on non-convex problems and can, in some cases, provide better solutions than those provided by existing nonlinear solvers.

Table of Contents

List of Tables xi

List of Figures xiii

Chapter 1: Introduction 1

Chapter 2: Piecewise Linear Approximation 8

- 2.1 Problem Reformulation 8
- 2.2 Grid Selection 9
- 2.3 MIP Formulation 10
  - 2.3.1 Lambda Model for One-Dimensional Function 10
  - 2.3.2 Logarithmic Lambda Model for Rectangular Grids 15
  - 2.3.3 Logarithmic Lambda Model for Union Jack Grid 17

Chapter 3: Literature Review 20

- 3.1 Functional Decomposition 20
- 3.2 Grid Design 20
- 3.3 MIP Modeling 23
- 3.4 Global Optimization 25

Chapter 4: Improved Grid Design 27

- 4.1 Effect of Grid Shape 27
  - 4.1.1 Union Jack Grid 27
  - 4.1.2 Crisscross Grid 29
- 4.2 Effect of Relative Cell Size 32
  - 4.2.1 One-Dimensional Shortest Path-based Method 35
  - 4.2.2. Alternating Shortest Path-based Method 38

Chapter 5: MIP Model Strengthening 42

- 5.1 Problem Strengthening Using Shared Grids 42
  - 5.2.1 Individual Grid Models 43
    - 5.2.1.1 Individual Grid Model with Segment-wise Adjacency 44

5.2.1.2 Individual Grid Model with Cumulative Adjacency 47
5.2.1.3 Individual Grid Model with Logarithmic Indexing 49
5.2.1.4 Individual Grid Model with SOS-2-based adjacency 50
5.2.2 Pattern-based Models 51
5.2.2.1 Pattern-based Model with Segment-wise Adjacency 57
5.2.2.2 Pattern-based Model with Cumulative Adjacency 59
5.2.2.3 Pattern-based Model with Logarithmic Indexing 60
5.2.2.4 Pattern-based model with SOS-2-based adjacency.........61
5.2.3 Combined Partition Models 62
5.2.3.1 Combined Partition Model with Segment-wise Adjacency64
5.2.3.2 Combined Partition Model with Cumulative Adjacency67
5.2.3.3 Combined Partition Model with Logarithmic Indexing..69
5.2.3.4 Combined Partition Model with SOS-2-based Adjacency70
5.2 Problem Strengthening using Constraint-based Inequalities 71
5.2.1 Upper Bound on a Variable Inducing an Upper Bound on another Variable..73
5.2.2 Upper Bound on a Variable Inducing a Lower Bound on another Variable..75
5.2.3 Lower Bound on a Variable Inducing an Upper Bound on another Variable..77
5.2.4 Lower Bound on a Variable Inducing a Lower Bound on another Variable..79
5.3 Problem Strengthening Using Bounds...81
5.3.1 Problem Strengthening Using Variable Bounds 81
5.3.2 Problem Strengthening Using Constraint-based Bounds...........86
Chapter 6: Computational Results 88
6.1 Goals 88
6.2 Introduction..91
6.2.1 Hardware and Software Specifications......................................91
6.2.2 Metrics for Measuring MIP Solution Quality............................92
6.2.3 Reformulation Guidelines...94

6.2.3.1 Minimizing Number of Grids 94
6.2.3.2 Applying Periodic Transformations...............................96
6.2.3.3 Scaling Variables 97
6.3 Applying the Basic PLA Method..100
6.3.1 Baseline Results for Ali et al. Problems 100
6.3.2 Baseline Results for Pooling Problems....................................107
6.3.3 Baseline Results for Global-Lib Problems 108
6.3.4 Baseline Results for MINLP-Lib Problems..............................111
6.4 Effect of Increased Grid Resolution 112
6.4.1 Effect of Increased Grid Resolution on Ali et al. Problems 113
6.4.2 Effect of Increased Grid Resolution on Global-Lib Problems..116
6.4.3 Effect of Increased Grid Resolution on MINLP-Lib Problems 118
6.5 Effect of Non-uniform Grids 120
6.6 Effect of Grid Shape 126
6.7 Effect of Indexing Scheme..128
6.8 Comparison With CONOPT 129
6.9 Comparing Grid-based and Pattern-based Models 133
6.10 Comparing Pattern-based and Combined Partition Models...............135
6.11 Effect of Problem Reduction and Problem Strengthening Techniques136
6.12 Comparison with DICOPT 139

Chapter 7: Conclusions 141

Appendix A1: Proof of Proposition 1 143

Appendix A2: Algorithm for One-dimensional Shortest Path-based Method.....145

Appendix A3: Alternating Shortest Path-based Method for Crisscross grids 151

Appendix A4: Alternating Shortest Path-based Method for Union Jack Grids...162

Appendix A5: Marginal Weight Consistency Constraints for a Combined Partition167

Appendix A6: Validity of Constraint-based Inequalities....................................170

Appendix A7: Selected Functional Decompositions of Ali et al. Problems........187

Appendix A8: Determining the Best Non-uniform Grid for an *n*-dimensional Function 213

Appendix A9: Models and Reformulations for Gasnet 220

Appendix A10: Model and Reformulation for Reactor Network Design............228

Appendix A11: LP Equivalence of Segment-wise and Cumulative adjacency...231

Appendix A12: Notation for MIP Models 233

References 235

List of Tables

Table 2.1: Standard binary indices....13
Table 2.2: Reflected binary indices 13
Table 6.1: Computational design 90
Table 6.2: Baseline results for Ali et al. problems....101
Table 6.3: Results for poor reformulations of selected Ali et al. problems 104
Table 6.4: Data for Hartman 3 105
Table 6.5: Baseline results for pooling problems 107
Table 6.6: Baseline results for Global-Lib problems....108
Table 6.7: Baseline results for MINLP-Lib problems 111
Table 6.8: Effect of increased grid resolution on Ali et al. problems....113
Table 6.9: Effect of increased grid resolution on Global-Lib problems 117
Table 6.10: Effect of increased grid resolution on MINLP-Lib problems 119
Table 6.11: Effect of using non-uniform grids 123
Table 6.12: Benchmarking grids generated by the shortest-path-based method .126
Table 6.13: Effect of using different grid shapes....127
Table 6.14: Effect of using different indexing schemes 128
Table 6.15: Ali et al. Problems that could not be solved by CONOPT 130
Table 6.16: Global-Lib problems that could not be solved by CONOPT 131
Table 6.17: Applying CONOPT to pooling problems....132
Table 6.18: Comparing grid-based and pattern-based models....133
Table 6.19: Comparing pattern-based and combined partition models 135
Table 6.20: Effect of problem strengthening strategies 137
Table 6.21: MINLP-Lib problems that could not be solved by DICOPT 139

Table A7.1: Data for Hartman 3 192

Table A7.2: Parameter set I for Hartman 6...193

Table A7.3: Parameter set II for Hartman 6 193

Table A7.4: Data for the Kowalik Problem...195

Table A7.5: Data for Meyer and Roth Problem...197

Table A7.6: Data for Modified Langerman Problem...198

Table A7.7: Data for Price Transistor..204

Table A7.8: Data for Shekel Problems 207

List of Figures

Figure 2.1: Common triangular grids..10

Figure 2.2: Approximating a function by a piecewise linear function 11

Figure 2.3: Indexing of a one-dimensional grid using binary indices 14

Figure 2.4: Indexing of a one-dimensional grid using reflected binary codes.......15

Figure 2.5: Binary indexing for a Union Jack grid 18

Figure 3.1: Common triangulations 22

Figure 4.1: Cell Orientation for K and Union Jack triangulations.........................28

Figure 4.2: Cell orientation in a Crisscross triangulation 30

Figure 4.3: Indexing scheme for a Crisscross triangulation 31

Figure 4.4: Example of a uniform grid versus a non-uniform grid........................34

Figure 4.5: Building blocks of a Crisscross triangulation.....................................39

Figure 4.6: Computing approximation error for a Crisscross triangulation...........40

Figure 5.1: Domain partitioning for functions f_1 and f_2..52

Figure 5.2: LP solution that is not feasible for a *pattern-based* model.................56

Figure 5.3: Example of a combined partition 63

Figure 5.4: Relating break point indices in individual patterns and the combined partition..64

Figure 5.5: Comparing the strength of pattern-based and combined partition models 67

Figure 5.6: Upper bound on X inducing an upper bound on Y 73

Figure 5.7: Upper bound on X inducing a lower bound on Y 75

Figure 5.8: Lower bound on X inducing an upper bound on Y.............................77

Figure 5.9: Lower bound on X inducing a lower bound on Y...............................79

Figure 5.10: Bounding vertex weight variables using an external lower bound ...83

Figure 5.11: Bounding vertex weight variables using an external upper bound ...85

Figure 6.1: Non-uniform segments in Epistatic Michalewicz 121

Figure 6.2: Plot of two-dimensional exponential function in *ex8_3_1*...............124

Figure 6.3: Nonuniform grid for the exponential function in *ex8_3_1*...............125

Figure A1.1: Union Jack triangulation versus Crisscross triangulation 143

Figure A3.1: Procedures for alternating shortest-path-based method 152

Figure A3.2: Computing approximation error for a triangular pattern...............161

Figure A4.1: Orientations of the two dimensional Union Jack triangulation......162

Figure A4.2: Triangular patterns for a two dimensional Union Jack triangulation162

Figure A4.3: Vertical and horizontal patterns for a Union Jack triangulation.....163

Figure A4.4: Approximation error for a break point pair along horizontal axis..164

Figure A4.5: Approximation error for break point pair along the vertical axis...164

Figure A4.6: Alternating shortest-path-based method for Union Jack grid.........166

Figure A5.1: Combined partition...168

Figure A6.1: Upper bound on X inducing upper bound on Y.............................171

Figure A6.2: Scenarios when upper bound on X induces an upper bound on Y..173

Figure A6.3 Upper bound on X inducing a lower bound on Y............................175

Figure A6.4: Scenarios when upper bound on X induces a lower bound on Y....177

Figure A6.5: Lower bound on X inducing an upper bound on Y........................179

Figure A6.6: Scenarios when lower bound on X induces an upper bound on Y..181

Figure A6.7: Lower bound on X inducing a lower bound on Y..........................183

Figure A6.8: Scenarios when lower bound on X induces a lower bound on Y....185

Chapter 1: Introduction

Research on nonlinear optimization has made great strides in the last few decades. Most of these problems are non-convex, which implies that local solvers, which start at an initial point and move to the nearest local optimum, find it difficult to solve these problems effectively. Global optimization seeks to find globally optimal solutions to optimization problems. In this dissertation, we study the effectiveness of solving non-linear programs by approximating the non-linear functions in the problem with piecewise linear functions, thereby creating an approximate mixed integer program (MIP), which can be solved by a MIP solver. The *piecewise linear approximation* (PLA) approach involves using the MIP solution as a starting point for a local solver. If the MIP solution lies in the basin of attraction of the globally optimal solution, then a local solver will yield the globally optimal solution.

Nonlinear optimization problems appear in many contexts in operations management, economics, and engineering. Hydropower generation (GonzAlez and Castro 2001) involves storing water in a reservoir and then converting the potential energy of the water into electrical energy by moving the water through a turbine. The power produced by the turbine is a function of (a) the quantity of water flowing through it, and (b) the height or head of the water column. Thus, to create a generation schedule for the turbine, both the volume and the height of the water have to be considered.

Oil refineries buy crude oils of different grades and mix them in mixing tanks to obtain different blends of gasoline. The quality of the blends and the crude oils is measured in terms of attributes such as octane number, density, etc. When two or more crude oils are mixed, the attribute value of the resulting blend is a weighted sum of the attribute values of the input crude oils, where the weights are the proportions of the crude oils mixed. These blends are sold as different products each with a specified selling price and with a specified minimum or maximum level of the various attributes. This gives rise to the pooling problem (Adhya et al. 1999) that involves determining the proportions in which the input crude oils should be mixed so that the total profit is maximized. Scheduling operations at an oil refinery is also a mixed integer nonlinear optimization problem (Karuppiah et al. 2008).

Nonlinear optimization problems also arise in the optimization of oil extraction process. Oil companies extract oil from a reservoir by injecting water into a reservoir through a set of wells called *injectors*. This injection process forces a mixture of water and oil to come out of another set of wells called *producers*. The quantity of mixture coming out of the producer wells depends upon a) the injection schedule, b) the connectivity between the wells, and c) the response delay (compressibility) of the producer well. To estimate the connectivity and the response delay, non-linear regression models (Yousef et al. 2006) are applied to historical injection and production data. Correct estimation of these characteristics helps determine an oil production schedule that maximizes total profit over the life span of the reservoir.

The Euclidean Multi-facility Location problem (Radó 1988) involves finding the location (Euclidean co-ordinates) of a set of plants that supply a single commodity to a set of demand points with pre-specified Euclidean coordinates and demand. Each plant has a specified maximum production capacity. The cost of transporting a specific quantity of the commodity from a plant to a demand point is the product of the quantity transported and the Euclidean distance between the plant and the demand point. The goal is to find the geographical positions of the plants so as to minimize the total transportation cost.

Nonlinear convex problems like the ones described above have traditionally been solved by local solvers which given a starting point, search for a descent direction (for a minimization problem), choose a step size and then move to a better point. This sequence of steps is repeated until the algorithm reaches a point at which it cannot find a descent direction, i.e., it reaches a locally optimal solution. However, for a non-convex problem, this solution need not be the globally optimal solution. *Multi-start algorithms* try to get the globally optimal solution by applying the local search procedure to multiple starting points and choosing the best among all the locally optimal solutions obtained. However, different starting points often lead to the same locally optimal solution. We say that these starting points lie in the *basin of attraction* of the locally optimal solution. Piecewise linear approximation seeks to generate 'good' starting points, preferably ones that lie in the basin of attraction of the globally optimal solution. In this approach, we approximate the non-linear functions in the original objective function or constraints by piecewise linear functions defined over the vertices of a grid that partitions the domain of each nonlinear function into cells. Using an appropriate mixed integer programming model, we approximate the original nonlinear program with a mixed integer program (MIP) whose solution serves as a starting point for a local nonlinear solver. In this dissertation, we validate the effectiveness of the PLA approach and propose various algorithmic strategies to improve its performance.

Applying the piecewise linear approximation approach to a general nonlinear problem requires the following four steps.

1. Function Decomposition: Depending upon the dimensionality of the grids that we wish to use in the piecewise linear approximation, we can express each nonlinear function within the optimization problem as a composition of other nonlinear functions, each of which has no more than a specific dimension. This transformation generally requires adding more variables and constraints to the original non-linear program. For example, if the original problem contains a nonlinear function $f(x_1, x_2, x_3)$ then we can either express f as a three dimensional function or we can create an intermediate function $y = g(x_2, x_3)$, express f as a composition of $g(x_2, x_3)$, i.e., $f(x_1, x_2, x_3) = f(x_1, g(x_2, x_3))$ and add a new constraint $y = g(x_2, x_3)$ to the nonlinear program.

2. Grid Design: Given the decomposition of the nonlinear problem into appropriate lower dimensional functions, we approximate each function using a grid that partitions the domain of that function into cells such that the function value at any point within a cell is approximated by a convex combination of the function values at the vertices of the cell. Grid design refers to the design (i.e, the shape and size) of the cells of the grid.

3. MIP Modeling: After choosing the functions and the grids for approximating the functions, we choose a mixed integer programming model to express the piecewise linear function approximation as a mixed integer program and thereby convert the original nonlinear program into an approximate mixed integer program. We then solve this mixed integer program (to some pre-specified gap) using an MIP solver.

4. Local solution: Given the solution to the approximate MIP, we solve the original nonlinear program with a local nonlinear solver which uses the MIP solution as a starting

point. For a Mixed Integer Nonlinear Program (MIINLP), we may choose to fix the value of the integer variables in the original nonlinear problem to their values in the MIP solution.

In our study, we apply the basic PLA approach to a diverse set of nonlinear problems and develop various strategies that improve one or more of the four steps mentioned above and thereby improve the effectiveness of the approach. We measure the performance of the PLA approach in terms of both the time it takes to get a solution and how close the PLA-based solution is to the globally optimal solution. The key contributions of this research are as follows.

1. Although piecewise linear approximation of non-linear functions is not new, no one has systematically studied the effectiveness of this approach as a global optimization strategy. Using a diverse set of over 140 continuous and discrete nonlinear optimization problems used as benchmarks by researchers, we show that the PLA approach is a promising solution methodology.

2. We examine how the performance of the PLA approach is influenced by the design of the grid used to create the approximation. We look at two aspects of grid design: cell shape and cell size.

a. Cell shape: We evaluate how rectangular and triangular grids perform on various problems. Within triangular grids, we examine the performance of different triangulations such as Union Jack and Crisscross. We also prove that for the bilinear function ($z = x_1x_2$), the Crisscross grid gives a smaller and stronger mixed integer program than that obtained using a Union Jack grid.

3. Cell size: The cell size, which determines the grid resolution, affects the size of the resulting mixed integer program. A small cell size or a high grid resolution increases the chance of obtaining a MIP solution that is close to the globally optimal solution. However, a high grid resolution also increases the size (and very often the solution time) of the resulting MIP. A carefully designed non-uniform grid (with cells of different sizes) can lead to a grid with a high approximation quality but a small MIP size. We propose an algorithm to create good non-uniform grids in two dimensions by extending a shortest path-based method already known in literature.

4. We examine how the performance of the PLA approach depends upon function decomposition and provide guidelines on how to obtain good reformulations of nonlinear programs. For example, we show how function decomposition can affect the size of the mixed integer program and thereby determine whether or not we can solve it within a reasonable time limit.

5. We propose stronger mixed integer programming models for applying the PLA approach to nonlinear programs. The traditional PLA approach separately linearizes each nonlinear function in a nonlinear program. However, if the same variable appears in multiple functions, then we can exploit the fact that in the MIP solution, the variable will take the same value across all the grids. This observation leads us propose smaller and stronger mixed integer programming models for the PLA approach.

6. We propose problem strengthening strategies to improve the MIP solution process, which determines the effectiveness of the overall approach.

a. Constraint-based valid inequalities: For those nonlinear programs in which the variables present in the nonlinear functions are related by constraints, we propose valid inequalities that use the constraint information to improve the LP relaxation values of the PLA-based mixed integer program and help accelerate the branch and bound procedure used for solving mixed integer programs.

b. Problem reduction strategies: Many nonlinear problems have pre-specified bounds on the values of the dependent variables of nonlinear functions. When we approximate these nonlinear functions, then we can use these bounds to exclude certain regions of the domain of the function, which translates into removing variables from the mixed integer program associated with the linearization. This process results in a smaller and a stronger mixed integer program.

7. The PLA approach extends naturally to mixed integer nonlinear programs (MINLP's) which is one of the hardest classes of problems. We analyze the performance of PLA on a modest set of small to medium-sized MINLP test problems.

8. Finally, we benchmark the performance of the PLA approach against some of the widely used local NLP solvers such as CONOPT and local MINLP solvers such as DICOPT. We found that the PLA approach can lead to better solutions than those obtained by solvers such as CONOPT and DICOPT.

The structure of this dissertation is as follows. In Chapter 2, we discuss how to apply the PLA method to a general nonlinear program. In Chapter 3, we review the literature related to different aspects of piecewise linear approximation. We discuss our methods for improved grid design in Chapter 4 and strategies for strengthening the PLA-based MIP models in Chapter 5. We present the computational results in Chapter 6 and conclude in Chaper 7.

Chapter 2: Piecewise Linear Approximation

In this chapter, we discuss how to solve a general nonlinear program using the Piecewise Linear Approximation (PLA) approach. Applying the PLA method to a general nonlinear program entails four steps—problem reformulation, grid selection, MIP creation and application of a local nonlinear solver. We now discuss these methods in more detail.

2.1 Problem Reformulation

In the first step, we identify within the non-linear program those non-linear functions whose dimension is greater than a specific value. We recursively express these functions as compositions of lower dimensional functions until each function in the nonlinear program has no more than a specific dimension. This generally requires adding new variables and new equality constraints to the original nonlinear program such that each equality constraint corresponds to a composition of functions in the original nonlinear program. Recasting the nonlinear program in a higher dimensional space (because of the addtition of new variables) is called *reformulation*.

There might be multiple ways of reformulating a non-linear program and the effectiveness of the piecewise linear approximation approach will depend upon the manner in which the non-linear functions are reformulated. For example, the non-linear program $\text{Min } y = \frac{x_1 x_2}{x_3},\ x_1, x_2 \in [10,20],\ x_3 \in [1,2]$ can be reformulated in two ways.

<u>Reformulation R1</u> $Min\, y$ such that $y = \frac{w}{x_3}, w = x_1 x_2$

$$x_1, x_2 \in [10, 20],\ w \in [100, 200],\ x_3 \in [1, 2]$$

Reformulation R2

$$Min\ y \text{ such that } y = ux_{2,}\ u = \frac{x_1}{x_3}$$

$$x_1, x_2 \in [10, 20], u \in [5, 20], x_3 \in [1,2]$$

For the same grid granularity, these two reformulations would differ in terms of their approximation qualities and the qualities of the MIP solutions since the reformulations involve different types of functions. Reformulation R2 might be preferred to reformulation R1 since the domain of the *u* variable is much smaller than that of the *w* variable which might result in better approximation quality for the grid that linearizes $y = \frac{w}{x_3}$ than for the one that linearizes $y = ux_2$.

2.2 Grid Selection

The second step in applying the PLA approach is to decide the shape and size of the grid used to linearize each non-linear function. Grids can be of any shape—rectangular, triangular, octagonal, etc. Further, triangular grids can again have different shapes as shown in Figure 2.1. The advantage of a triangular grid is that each point in the function domain can be uniquely expressed as a convex combination of the grid points, i.e., of the vertices of the cells of the grid.

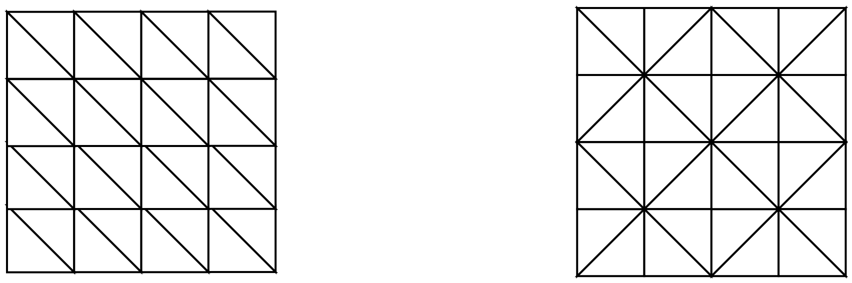


Figure 2.1: Common triangular grids.

2.3 MIP Formulation

The third step in the linearization process entails formulating a mixed integer program. Here we can choose from a variety of MIP formulations (Vielma et al. 2014). Each formulation ensures that for each point in the domain, we select the cell in which that point lies and express the function value at that point as a convex combination of the function value at the vertices of the chosen cell. We now review the one and two-dimensional *lambda-models* (Vielma et al. 2010, Croxton et al. 2003).

2.3.1 Lambda Model for One-Dimensional Function

Given a function $f(X):[l,u]\rightarrow \mathbf{R}$, we choose a partition $l=x_0<x_1<...<x_n=u$ to divide the domain of f into n segments using n + 1 break points. The piecewise linear approximation is defined by n pairs of real numbers $(m_i,\, c_i)$ such that

$$f(X)=m_iX+c_i, \quad X\in[x_{i-1},x_i] \quad \text{for all } i=1,...,n \tag{2.1}$$

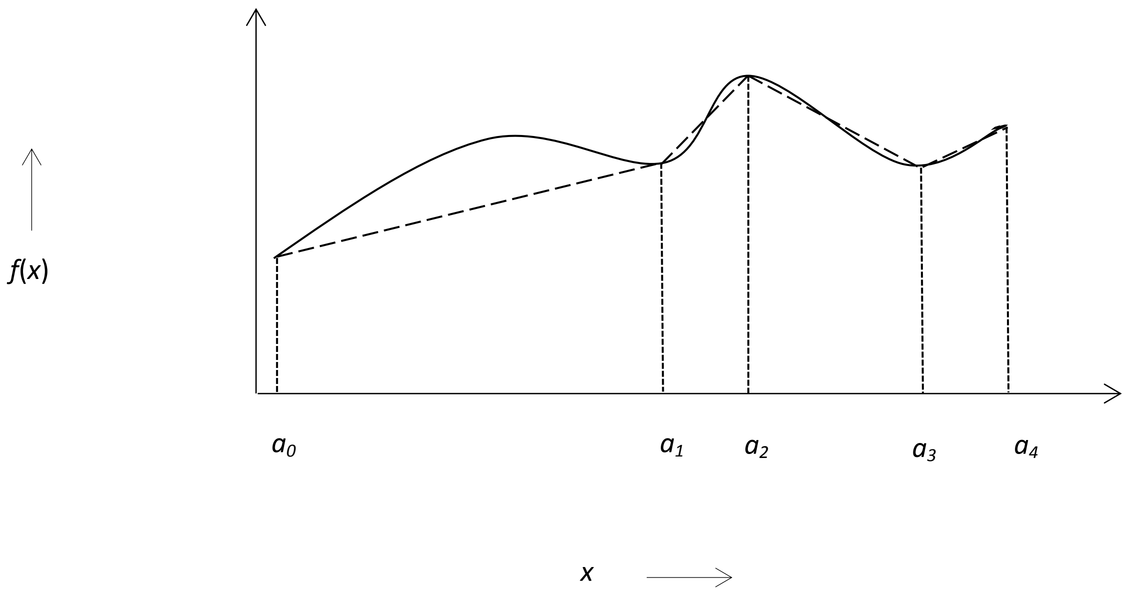


Figure 2.2: Approximating a function by a piecewise linear function

We model this problem as a mixed integer program in which we associate a continuous variable λ to each break point, a binary variable Z to each segment and then add constraints that ensure that the variable value and the function value at a point is given by a convex combination of the variable values and function values at the end points of the segment in which that point lies. The *lambda* or the convex combination model is described by Equations (2.2–2.7) (Croxton et al 2003).

$$X = \sum_{i=0}^{n} x_i \lambda_i \tag{2.2}$$

$$f(X) = \sum_{i=0}^{n} \lambda_i f(x_i) \tag{2.3}$$

$$\lambda_{i-1} + \lambda_i \geq Z_i \quad \text{for all } i = 1, 2, ..., n \tag{2.4}$$

$$\sum_{i=1}^{n} Z_i = 1 \tag{2.5}$$

$$\sum_{i=0}^{n} \lambda_i = 1 \tag{2.6}$$

$$\lambda_i \geq 0 \quad \text{for all } i = 0,\ldots,n \tag{2.7}$$

$$Z_i \in \{0,1\} \quad \text{for all } i = 1,2,\ldots,n \tag{2.8}$$

The basic lambda model can be improved further by observing that we can identify n by using just $\log_2(n)$ bits of information. As shown by Vielma and Nemhauser (2008, 2009), we can have an indexing scheme with $\log_2(n)$ binary variables such that each segment is assigned a binary index comprised of $ceil(\log_2(n))$ digits. Subsequently, they add two constraints for each digit, each constraint forcing a set of interval end points to have a weight of zero. These constraints ensure that the lambda variables for exactly one interval can have a positive value. We now describe the logarithmic lambda model.

Let $\bar{I}(q)$ be the set of break points each of which serves as the end point of a segment whose binary index has a zero at the q'th position (from the left), and $\hat{I}(q)$ be the set of break points each of which serves as the end point of a segment whose binary index has a one at the q'th position. The *logarithmic lambda* model can then be described by Equations (2.2–2.3), Equations (2.6–2.7) and Equations (2.8–2.9).

$$\sum_{i \notin \bar{I}(q)} \lambda_i \leq B_q \quad \text{for all } q = 1,2,\ldots,\log_2(n) \tag{2.8a}$$

$$\sum_{i \notin \hat{I}(q)} \lambda_i \leq 1 - B_q \quad \text{for all } q = 1,2,\ldots,\log_2(n) \tag{2.8b}$$

$$B_q \in \{0,1\} \quad \text{for all } q = 1,2,\ldots,\log_2(n) \tag{2.9}$$

However, a naïve binary indexing scheme in which we index the intervals sequentially by expressing their base-10 index in base-2 format will not work here. Say we have 9 grid points and 8 cells that we index using 3 binary digits as shown in Table 2.1 and Figure 2.3.

Cell	**Binary Index**	**Cell**	**Binary Index**
Cell 1	000	Cell 5	100
Cell 2	001	Cell 6	101
Cell 3	010	Cell 7	110
Cell 4	011	Cell 8	111

Table 2.1: Standard binary indices

Suppose the *MIP* chooses a point in Interval 1 with binary representation '000'. In this case, Equations (2.8) yield the following.

1. $\lambda_6 + \lambda_7 + \lambda_8 + \lambda_9 \leq 0$ (since Digit 1 or B_1 is equal to 0)
2. $\lambda_4 + \lambda_8 + \lambda_9 \leq 0$ (since Digit 2 or B_2 is equal to 0)
3. $\lambda_9 \leq 0$ (since Digit 3 or B_3 is equal to 0)

Cell	**Binary Index**	**Cell**	**Binary Index**
Cell 1	000	Cell 5	110
Cell 2	001	Cell 6	111
Cell 3	011	Cell 7	101
Cell 4	010	Cell 8	100

Table 2.2: Reflected binary indices

Thus, λ_4, λ_6, λ_7, λ_8 and λ_9 are set to zero, which implies λ_1, λ_2, λ_3 and λ_5 can all have positive weights. However, the adjacency condition requires that if interval 1 is selected, then only $\lambda_1$and λ_2 can have positive weights. Thus, as Vielma and Nemhauser (2011) show, a naïve binary indexing scheme cannot be used. To resolve this issue, Vielma and Nemhauser (2011) proposed that the indexing of the cells should be such that adjacent cells differ in their representation by only one bit. Such a binary code is called a *reflected binary code* or a *gray code*. One such indexing scheme is described in Table 2.2 and displayed in Figure 2.4. We can check that these binary codes let the *MIP* satisfy the adjacency conditions for each cell.

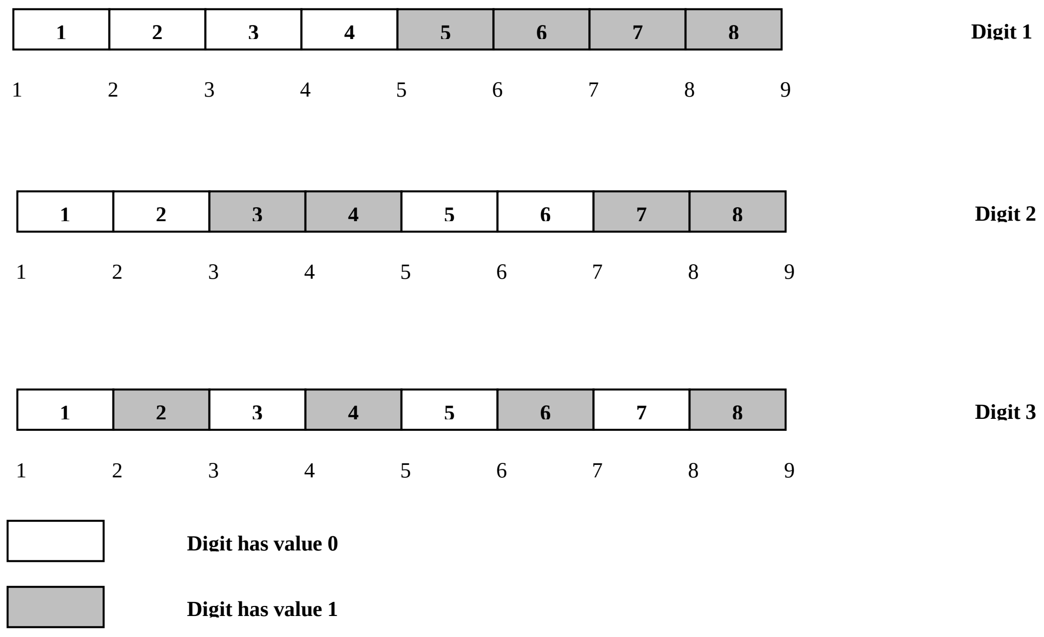


Figure 2.3: Indexing of a one-dimensional grid using binary indices

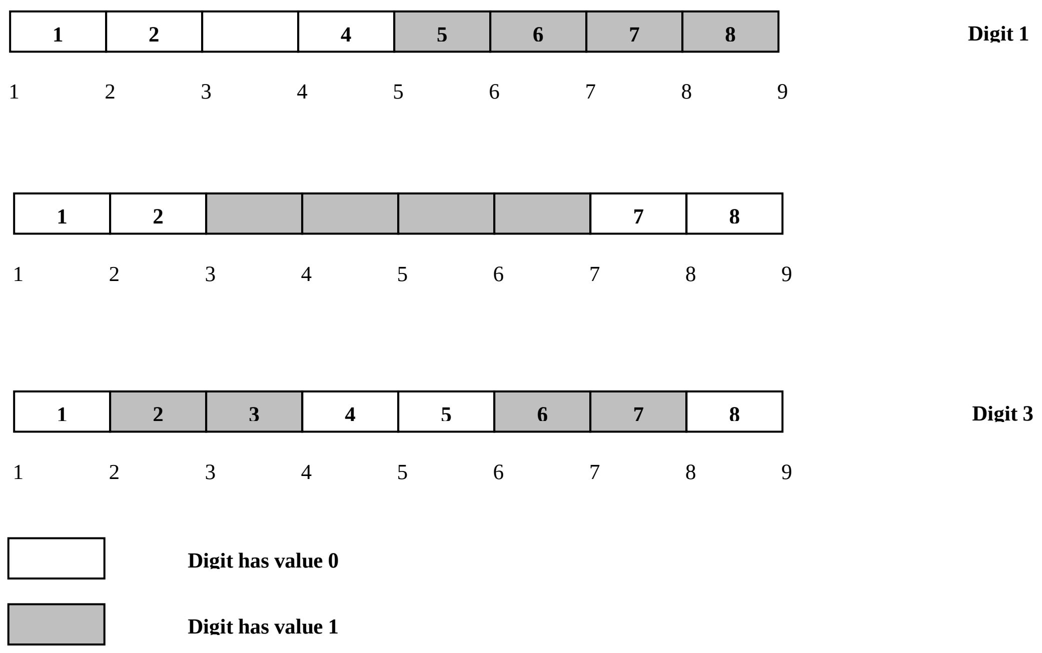


Figure 2.4: Indexing of a one-dimensional grid using reflected binary codes

2.3.2 Logarithmic Lambda Model for Rectangular Grids

We now discuss the *logarithmic lambda model* for rectangular grids. Let us assume that we are trying to approximate the function $f(x, y)$ by using a rectangular grid. For a rectangular grid, we can uniquely identify a cell by identifying the segments that the cell corresponds to along the X and Y axes. Consider a rectangular grid with m segments along the X-axis and n segments along the Y-axis. To identify a cell, we need $\log_2 m$ digits to identify the X-segment and $\log_2 n$ digits to identify the Y-segment corresponding to the cell. Thus, we need '$\log_2 m + \log_2 n$' binary variables in the mixed integer program, which we now describe.

Notation

- V: set of vertices
- v: index over V
- n_X : number of segments along the X-axis
- n_Y : number of segments along the Y-axis
- k: index over the digits of the reflected binary code for segments along the X-axis
- l: index over the digits of the reflected binary code for segments along the Y-axis
- x_v : X-value of vertex v
- y_v : Y-value of vertex v
- f_v : function value for vertex v
- $V_X^0(k)$: set of vertices that correspond to an X-segment whose reflected binary code has a zero at the k'th position
- $V_X^1(k)$: set of vertices that correspond to an X-segment whose reflected binary code has a one at the k'th position
- $V_Y^0(l)$: set of vertices that correspond to a Y-segment whose reflected binary code has a zero at the l'th position
- $V_Y^1(l)$: set of vertices that correspond to a Y-segment whose reflected binary code has a one at the l'th position
- $\hat{h}$: approximate function value for f

Variables

- α_v : continuous variable that denotes the weight assigned to vertex v
- B_k^X : binary variable that is one if we select an X-segment whose reflected binary code has a one at the k'th position

- B_l^Y : binary variable that is one if we select a *Y*-segment whose reflected binary code has a one at the *l*'th position

<u>Constraints</u>

$$\hat{h} = \sum_{v \in V} \alpha_v f_v \tag{2.10}$$

$$x = \sum_{v \in V} \alpha_v x_v \tag{2.11}$$

$$y = \sum_{v \in V} \alpha_v y_v \tag{2.12}$$

$$\sum_{v \in V} \alpha_v = 1 \tag{2.13}$$

$$\sum_{v \notin V_X^0(k)} \alpha_v \quad \leq \quad B_k^X \qquad \text{for all } k = 1, \ldots, \log_2(n_X) \tag{2.14}$$

$$\sum_{v \notin V_X^1(k)} \alpha_v \quad \leq \quad 1 - B_k^X \qquad \text{for all } k = 1, \ldots, \log_2(n_X) \tag{2.15}$$

$$\sum_{v \notin V_Y^0(l)} \alpha_v \quad \leq \quad B_l^Y \qquad \text{for all } l = 1, \ldots, \log_2(n_Y) \tag{2.14}$$

$$\sum_{v \notin V_Y^1(l)} \alpha_v \quad \leq \quad 1 - B_l^Y \qquad \text{for all } l = 1, \ldots, \log_2(n_Y) \tag{2.15}$$

$$B_k^X \in \{0,1\} \qquad \text{for all } k = 1, \ldots, \log_2(n_X) \tag{2.18}$$

$$B_l^Y \in \{0,1\} \qquad \text{for all } l = 1, \ldots, \log_2(n_Y) \tag{2.19}$$

$$\alpha_v \geq 0 \qquad \text{for all } v \in V \tag{2.20}$$

2.3.3 Logarithmic Lambda Model for Union Jack Grid

We now discuss the logarithmic lambda model for a triangular Union Jack grid. For such a grid with n_X segments along the *X*-axis and n_Y segments along the *Y*-axis, we need '$\log_2 n_X$ + $\log_2 n_Y$' bits to identify a pair of adjacent triangles belonging to a specific segment combination along the *X* and *Y* axes. To differentiate between the two cells in this pair, we shade one of the cells as gray, which corresponds to a binary index of zero, and the other as white, which corresponds to a binary index of one. Thus, we can identify each cell uniquely by '$\log_2 n_X + \log_2 n_Y + 1$' digits (Vielma 2011).

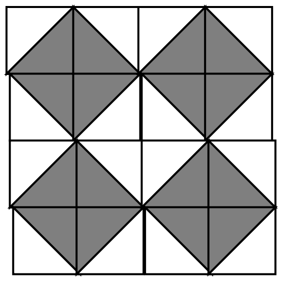


Figure 2.5: Binary indexing for a Union Jack grid

To obtain the model formulation for the Union Jack grid, we define an additional binary variable Q_{XY} that takes a value zero if we select a gray cell and a value one if we select a white cell. Let V_{XY}^0 be the set of vertices that touch a gray cell, and V_{XY}^1 be the set of vertices that touch a white cell. Then, the logarithmic lambda model for the Union Jack grid is comprised of Equations (2.10–2.22).

$$\sum_{v \notin V_{XY}^0} \alpha_v \leq Q_{XY} \tag{2.21}$$

$$\sum_{v \notin V_{XY}^1} \alpha_v \leq 1 - Q_{XY} \tag{2.22}$$

Thus, we find that triangular grids are modeled on similar lines as rectangular grids except that they require additional binary variables. Further, the Union Jack triangulation leads to the smallest mixed integer program among all triangular grids. In the next chapter we will see how we can use another triangular that can do even better than the Union Jack triangulation in terms of problem size.

Chapter 3: Literature Review

The literature on piecewise linearization can be broadly divided into four areas: Functional decomposition, grid design, MIP modeling, and global optimization.

3.1 Functional Decomposition

Arnold (1963) shows that every continuous function of more than two variables can be expressed as a composition of finitely many continuous functions of two variables. This result is the basis of the functional decomposition technique that we use to express higher dimensional functions as compositions of lower dimensional functions.

McCormick (1976) gave a procedure for obtaining the global solution to a factorable nonlinear programming program which is a nonlinear program in which each nonlinear function can be expressed as a product of two univariate nonlinear functions by appropriately defining intermediate compositions of functions. For example, consider the function $f(x, y, z) = xyz$, which is a product of three variables. This function can be treated as a function of two variables by defining another function $g(x, y) = w=xy$ and then expressing $f()$ as $f(w, z) = wz$. The procedure uses the fact that if we know lower bounding convex functions and upper bounding concave functions for the two uni-variate nonlinear functions, then we can find a lower bounding convex function and an upper bounding concave function for the original nonlinear function.

3.2 Grid Design

Grid design was initially studied for numerically solving differential equations using Finite Element Methods in engineering and for obtaining fixed points of functions in economics (Talman and Laan 1980). A distinction is made in finite element analysis between a mesh and a grid. A mesh refers to any kind of partition of the domain whereas a grid corresponds to a partition in which the cells have the same shape and can be generated by knowing the location of the break points or segments along the boundary the domain. Frey and George (2000) provide a comprehensive survey on mesh generation methods for both structured and unstructured meshes. In a structured mesh, we discretize the boundary of the domain and use these discretizations to create the mesh. In an unstructured mesh, boundary discretization is not enough to determine the actual domain partition. Methods for unstructured mesh generation include spatial decomposition methods (such as Quadtree and Octree-based methods), advanced front methods (in which we construct the mesh element by element, starting from an initial front) and Delaunay-based methods in which given an initial set of grid points, we alternate between creating a Delaunay triangulation and adding new grid points. A triangulation is defined as a partition composed of triangles or simplices such that the intersection of any two triangles is either empty or the common side of those triangles (Talman and Laan 1980). We depict the shapes of some common triangulations in Figure 3.1.

Within the context of the PLA approach, Chien and Kuh (1977) used simplicial subdivision to linearize the behavior of nonlinear resistive networks that contain resistors whose resistance values change with a change in temperature or light. Al-Khayal and Falk (1983) used rectangular grids to solve problems involving bilinear functions. They employed a branch and bound framework in which piecewise linear under-estimators were used to obtain lower bounds. Linderoth (2005) partitioned the domain into triangles and obtained tighter under-estimators and solve a quadratically constrained quadratic program in a branch and bound algorithm.

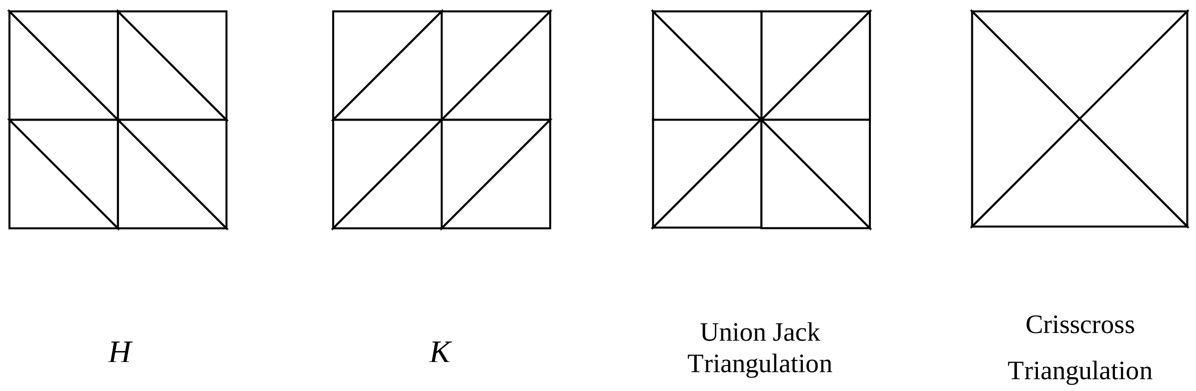


Figure 3.1: Common triangulations

Babayev (1997) used a *K*-triangulation (Lann 1980) to approximate a two dimensional non-linear function. However, this approach did not involve using the MIP solution as a starting point for a local nonlinear solver. González (2001) used the *H*-triangulation to solve a short-term hydro-electric generation problem in which the power produced by a turbine is a non-linear function of the water head (pressure) and the quantity of water discharged. Ambrosio et al. (2010) used a *K*-triangulation to solve the same hydro-generation problem. Misener and Floudas (2010) approximated a three dimensional function by using three-dimensional simplices. Meyer and Floudas (2004, 2005) used simplicial subdivision to develop convex envelopes for trilinear and edge concave functions. A function is said to be edge-concave if it is component-wise concave. Wang and Zhang (2008) solve a nonlinear program with linear constraints via piecewise linear approximation of the objective function over a simplicial partition. However, all these models used one binary variable per cell. Vielma and Nemhauser (2011) showed that we can develop a valid but smaller mixed integer programming model if we can assign a binary address (string of zeroes and ones) to each cell such that any two adjacent cells differ in their address by only one bit, i.e., the binary addresses correspond to a gray code (Gilbert 1957). Rovatti et al. (2014) proposed a (rectangular) grid-based piecewise linear approximation of nonlinear functions such that each point lying in the function domain is expressed as a convex combination of not just the vertices of the grid but also of a pre-specified set of sample points lying within the function domain. They found that such a representation gives the piecwise linear representation has more degrees of freedom and provides better computational performance. For a one dimensional grid, the design involves only the size of the cells (or segments). In this context, Magnanti and Stratila (2012) derived a bound on the number of segments required to approximate a one-dimensional concave function with piecewise linear functions such that a desired approximation quality is achieved.

3.3 MIP Modeling

When we approximate a nonlinear function with a mixed integer program, we are interested in MIP formulations that are sharp and locally ideal (Vielma 2014). Croxton et al. (2003) compared the three standard one dimensional MIP models for approximating non-linear functions with piecewise linear functions and showed that the three models—multiple choice model (Balakrishnan and Graves 1989), incremental or delta model (Dantzig 1960) and convex combination or lambda model (Dantzig 1960) — give the same LP relaxation value. These three formulations are sharp in that they provide the highest LP relaxation value as possible. Li and Yu (1999) proposed another MIP formulation in which binary variables are required only for those segments in which the nonlinear function is nonconvex. However, for a general non-convex function, this representation does not offer any advantage over the other representations. Li et al. (2009) proposed another two representations, one which used a Big-M and another that used a logarithmic number of binary variables. However, both representations, which Li et al. (2009) claimed to be superior to existing representations, are not sharp, i.e., they have a lower LP relaxation value than the three standard formulations. Vielma et al. (2010) present an LP solution for each of these representations and show that the LP relaxation value is lower than the LP relaxation value of the traditional models.

An MIP formulation is locally ideal if the ‘local’ LP formed by the constraints in the MIP formulation corresponding to the linearization of a specific nonlinear term (while omitting all other constraints in the nonlinear program) has integral corner points. Since there will be other variables and constraints in the model, a locally ideal formulation does not result in an MIP that has integral corner points. However, in general, solving a sharp model will take fewer branch and bound nodes than solving a model that is not sharp. Padberg (200) showed that the incremental model is locally ideal whereas the convex-combination model is not, though as Croxton et al. (2003) have shown, both these models are sharp. Sherali (2001) proposed a locally ideal dis-aggregate convex combination model with two continuous variables per grid point. Vielma et al. (2010) provided a unifying framework for creating piecewise linear representations of multidimensional nonlinear functions. They extend the one-dimensional models to *n*-dimensions and also discuss a logarithmic indexing scheme which results in fewer binary variables than the traditional models. Domschke et al. (2011) employed the locally ideal incremental model to linearize a two dimensional function appearing in a nonlinear program for solving a minimum cost flow problem over a gas network.

Keha et al. (2006) discuss how to enforce the adjacency conditions by defining appropriate variables as SOS-2 and solve a significantly smaller mixed integer program (with no binary variables) using specialized branching strategies that ensure that only adjacent grid points are selected. Vielma and Nemhauser (2011) report that in terms of solution times, these models are second only to the logarithmic models.

3.4 Global Optimization

Global optimization technniques aim at finding the optimal solution to optimization problems. Most of the deterministic global optimization techniques are of the type 'divide and conquer', a framework first used by Falk and Soland (1969). The algorithms in this category include the Branch and Reduce algorithm proposed by Ryoo and Sahinidis (1995), (1996), the α-Branch and Bound method proposed by Floudas et al (1995), the Branch and Contract algorithm proposed by Grossmann (1999), the Branch and Cut algorithm proposed by Kesavan and Barton (2000) and the Interval Analysis-based algorithm proposed by Vaidyanathan and El-Halwagi (1996). All these algorithms except the Interval Analysis-based algorithms require the computation of underestimating functions (convex envelopes) so as to obtain a lower bound on the function in a specific region of the function domain. Other deterministic algorithms for global optimization include the Extended Cutting Plane method proposed by Westerlund et al. (1998) and Westerlund and Pörn (2002), and Outer Approximation algorithm proposed by Duran and Grossmann (1986). Floudas and Gounaris (2009) provide a comprehensive survey on various global optimization techniques.

The main difference between the branch-and-bound based methods and the PLA approach is that the PLA approach does not require the computation of convex envelopes at every node in the branch and bound tree. As a result, the PLA approach can possibly have smaller solution times. However, the disadvantage is that the PLA method cannnot provide any information on the gap between the PLA solution and the globally optimal solution. One possible way of improving the PLA approach is to compute the convex envelopes for the partitions that are used in the linearization and thereby provide a gap between the objective value of the current PLA solution and thart of the global solution. If the gap is too high, then we can apply the PLA approach to the original problem again but with a more refined grid.

Chapter 4: Improved Grid Design

In this chapter, we discuss how different aspects of grid design influence the performance of the piecewise linear approximation approach. Grid design affects the performance of the piecewise linear approximation approach in two ways. First, it affects the quality of approximation of the nonlinear functions and second, it influences how fast the corresponding mixed integer program could be solved. For example, in a triangular grid, each point in the function domain has a unique representation in terms of the vertices of the triangle in which that domain point lies. On the other hand, for a rectangular grid, a domain point can have multiple convex combination representations. Therefore, a triangular grid will have better approximation quality than that of a rectangular grid. The design of the grid affects the size of the associated mixed integer program and therefore, the performance of the MIP solution procedure.

We focus on two aspects of grid design—cell shape and relative cell size. For cell shape, we discuss two types of grids—Union Jack and Crisscross—and show how for the bilinear function, a Crisscross grid can provide better approximation quality than a Union Jack grid. For relative cell size, we show how carefully designed non-uniform grids can provide good approximation quality without a large mixed integer program.

4.1 Effect of Grid Shape

4.1.1 Union Jack Grid

The shape of a grid affects the size of the mixed integer program corresponding to the piecewise linear approximation of a function. In a Union Jack triangulation, which has two triangular cells for each combination of horizontal and vertical segments, we uniquely identify a cell by using an ‘orientation’ flag (i.e., an additional binary variable) such that one cell has an orientation of ‘one’ and the other has an orientation of ‘zero’. In Figure 4.1, the shaded cells have an orientation value of one and the white cells have an orientation value of zero. This orientation flag (or binary variable) is then used in an adjacency constraint to ensure that we assign positive weights only to the vertices of the chosen cell. This constraint works as follows. If the binary variable (or the orientation flag) is zero, then we must choose a ‘white’ cell and therefore we cannot choose any grid point that does not touch a white cell and so any grid point that does not touch a white cell is assigned a vertex weight of zero. Similarly, if the binary variable (or the orientation flag) has value one, then we must choose a ‘gray’ cell and any grid point that does not touch a gray cell is assigned a weight of zero. The structure of the Union Jack grid helps us ensure adjacency by using just one variable for the entire grid in addition to the binary variables for the segments along the *X* and *Y* axes.

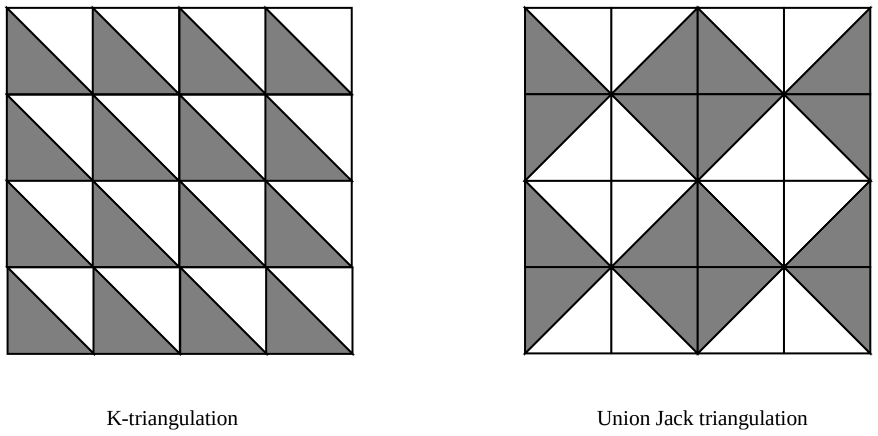


Figure 4.1: Cell Orientation for K and Union Jack triangulations

4.1.2 Crisscross Grid

Although the Union Jack grid leads to a parsimonious mixed integer program, we have found that the Crissscross triangulation (Wahlbin 1998) used in finite element analysis can provide an even smaller mixed integer program for the bilinear function that appears in a number of industrial problems such as pooling and water management problems (Faria et al. 2011).

Proposition1: For the bilinear function $f(x, y) = xy$, a Crisscross triangulation with uniform segments along each axis provides the same approximation quality as that provided by a Union Jack triangulation with twice the number of uniform segments along each axis. For proof, refer to Appendix A1.

To create a mixed integer program based on the Crisscross triangulation, we need to create a valid indexing scheme. To do so, we first need to ensure that the indexing satisfies the necessary 'Gray Code' condition. As per this condition, binary addresses of adjacent cells differ by exactly one bit. Let us assume that we have a Crisscross triangulation with one segment each along the two axes. This gives rise to four cells, as shown in Figure 4.2a. We seek an indexing scheme for these four cells such that neighboring cells differ by only one bit. To do so, we use the sequence '00', '01', '11' and '10' in which consecutive elements differ from one another in only one bit. This sequence can identify four cells using two binary digits, each of which can be considered as an orientation flag. We call these flags Orientation I (Figure 4.2a) and Orientation II (Figure 4.2b). For Orientation I, cells A and B have a flag value of 0 (shown as white) and cells C and D have a flag value of 1 (shown as gray). For Orientation II, cells B and

C have a flag value of 0 (shown as white) and cells A and D have a bit value of 1 (shown as gray). Thus, cells A, B, C and D have addresses ‘01’, ‘00’, ‘10’ and ‘11’ respectively.

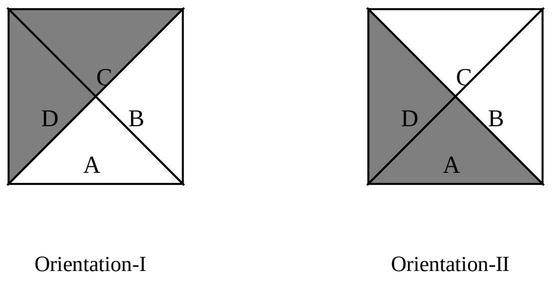


Figure 4.2: Cell orientation in a Crisscross triangulation

Using these patterns as the building blocks, we create a Crisscross triangulation (shown in Figure 4.3) such that (a) two adjacent cells that belong to the same horizontal segment and the same vertical segment differ in their addresses by exactly one bit, and (b) two adjacent cells that correspond to consecutive vertical segments or consecutive horizontal segments have the same bit value for both orientations. Thus, to identify a cell, we need to identify the horizontal and vertical segment to which that cell belongs and then identify the values of its Orientation I and Orientation II flags. This indexing scheme works because when we choose a specific triangular cell, we choose a set of orientations and these orientations prevent us from selecting any other vertex (except those of the chosen cell) belonging to the set of four cells that correspond to the same combination of horizontal and vertical segments.

This is similar to identifying a cell in a Union Jack triangulation in which we first identify the horizontal and vertical segments to which a cell belongs and then identify the value of its orientation flag. For example, to identify a cell in the 2x2 Crisscross triangulation shown in Figure 4.3 with a logarithmic indexing scheme, we need four bits—one bit for the horizontal segment (Figure 4.3b), one bit for the vertical segment (Figure 4.3c), one bit to identify the Orientation I flag (Figure 4.3d) and one bit to identify the Orientation II flag value for the cell (Figure 4.3e). On the other hand, if we use a Union Jack triangulation (with logarithmic indexing) that gives the same approximation quality, then we need to have 4 horizontal segments, 4 vertical segments and 5 bits—two for the horizontal segment, two for the vertical segment and one bit to identify the orientation.

In general, a Crisscross triangulation with *m* horizontal segments, *n* vertical segments (i.e., $4mn$ cells), a logarithm indexing scheme requires $\log_2(m) + \log_2(n) + 2$ bits.

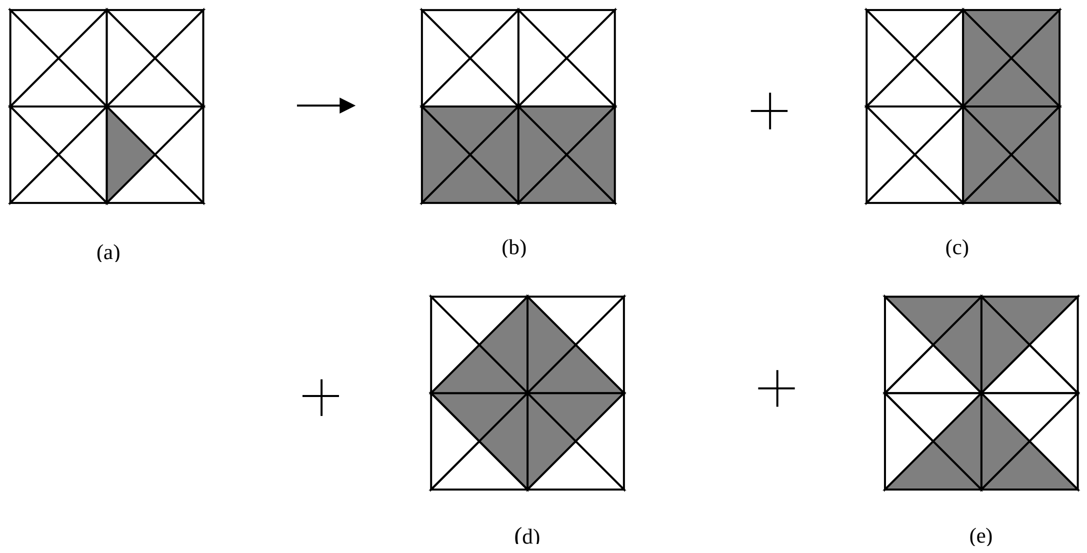
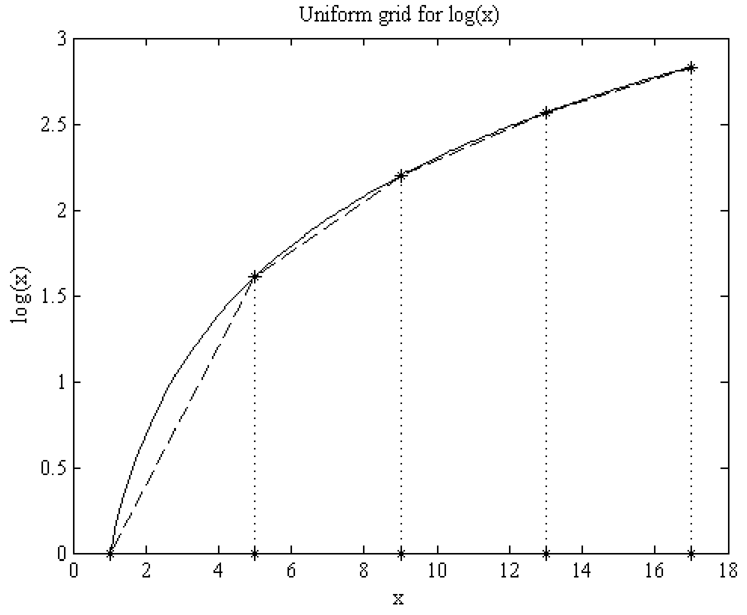


Figure 4.3: Indexing scheme for a Crisscross triangulation

4.2 Effect of Relative Cell Size

A smaller cell size implies a finer resolution and therefore a grid with high approximation quality in which for each point in the domain of the function, the difference between the true function value and the function value given by the linear approximation is small. By increasing the grid resolution, we increase the chances that the solution to the mixed integer program is close to the globally optimal solution to the original program. However, as we increase the grid resolution, the size of the mixed integer program increases, often making the program harder to solve. A good grid resolution is, therefore, the result of a trade-off between approximation quality and difficulty level of the mixed integer program.

We can have the benefit of a high resolution grid without the associated large mixed integer program by constructing a non-uniform grid that has high resolution only in those areas of the domain where the linearity of the function is small or equivalently where the curvature of the function is high. Although such non-uniform grids have been created by various mesh generation methods in finite element analysis (e.g., Frey and George 2000), they have not been used within the context of piecewise linear functions. To understand how a carefully-constructed non-uniform grid might be better than a uniform grid, let us approximate $f(x) = log(x)$, $x \in [1, 17]$, using two different piecewise linear approximations with 4 segments. In the first approximation, we place break points uniformly along the domain of the function, i.e., at $x = 1, 5, 9, 13$, and 17, whereas in the second, we use a non-uniform grid with break points at $x = 1, 3, 5, 9$, and 17. We observe that by placing the break points close to one another in regions with high curvature, the non-uniform grid provides a much better approximation quality than the uniform grid with the same number of break points (Figure 4.4). To create good nonlinear grids in one and two dimensions, we use a shortest path-based method. Although the one-dimensional version of this method is already known and used in the context of curve approximation (Dahl and Realfsen 2000), to the best of our knowledge, no one has actually used it to solve nonlinear programs within the PLA approach. In this section, we describe the one dimensional shortest path-based method and then propose a heuristic method that uses the same principle to generate good non-uniform grids in two dimensions.



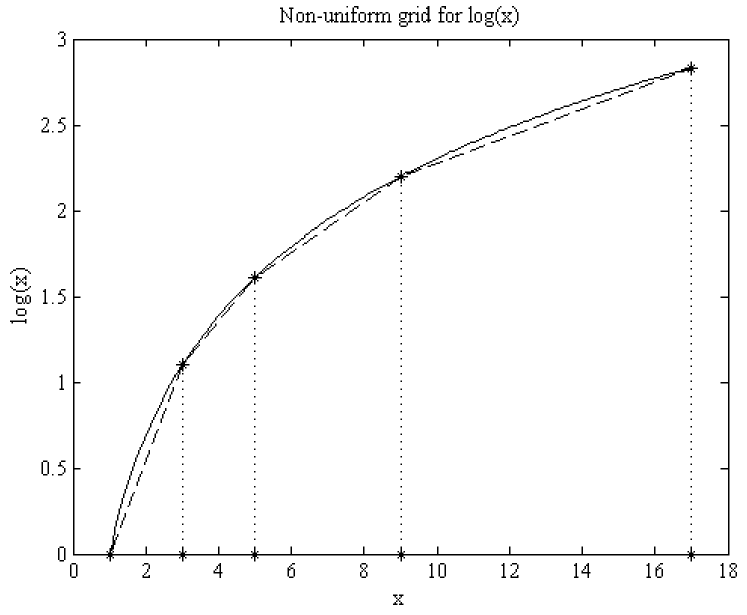


Figure 4.4: Example of a uniform grid versus a non-uniform grid

4.2.1 One-Dimensional Shortest Path-based Method

We employ a shortest path-based method to approximate a one-dimensional function with a piecewise linear function over a non-uniform grid. In this method, we divide the domain of the function into small non-overlapping intervals, create possible groups/aggregations of consecutive intervals called segments, find/estimate the approximation error for each segment and then choose a set of non-overlapping segments that cover the entire domain and give the best approximation in terms of a specific error metric.

The approximation error associated with a segment depends upon (a) the approximation errors that arise when the domain points lying within the segment are assigned an approximate function value given by a convex combination of the function values at the lower limit and upper limit of that segment, and (b) the chosen error metric, i.e., whether we are interested in the maximum or average of the absolute errors over the domain points lying within the segment. Our objective is to aggregate intervals into segments so that we get an approximation that minimizes a specific error metric while having no more than a pre-specified number of segments, which ensures that we do not have more than a specified number of binary variables in the resulting mixed integer program.

To determine the best way of aggregating intervals into segments, we use the shortest path method described in Dahl and Realfsen (2000), and Ahuja et al. (1993). Dahl and Realfsen (2000) provide the basic recursion used in this method. However, we operationalize the method and explain how to create a network from which we can obtain the shortest path. Ahuja et al. (1993) show how to approximate a piecewise linear function by choosing a subset of linear pieces. However, they do not put any limitation on the number of arcs or hops that are allowed in the path, something that we model in our algorithm. Our method entails creating a directed graph in which the nodes correspond to the interval end points and the arcs correspond to segments. The nodes are indexed in the order in which the corresponding interval end points appear along the domain of the function. The nodes corresponding to the lower and the upper limit of the function domain are called the *source* and *sink* respectively. For each pair of nodes, an arc is created from the lower-indexed node to the higher-indexed node and assigned a cost that depends upon the specified error metric and the approximation errors associated with a fixed set of randomly chosen sample points.

Given this network, any path from the source node to the sink node corresponds to a contiguous set of segments that cover the entire domain. Further, the cost of the path is a measure of the quality of the linear approximation. To ensure that we have the best linear approximation with no more than a given number of break points, we have to find the least cost path from the source node to the sink node such that the path contains no more than a specified number of arcs or hops. We call this problem the *hop constrained shortest path problem*. Here again we need to specify a metric for expressing the cost of a path as a function of the cost of the arcs lying on that path. For example, with a *sum of arc costs* metric, which yields the *min-sum path*, path cost is given by the sum of the costs of the arcs on that path. On the other hand, with a *maximum arc cost* metric, which gives the *min-max path*, the path cost is expressed as the cost of the highest-cost arc on that path. We now discuss this method more formally.

Consider a function $f(x)$ with domain $[l, u]$. We divide $[l, u]$ into n parts using break points $x_0, x_1, \ldots, x_n$ such that $l = x_0 < x_1 < \ldots < x_n = u$. Each pair of break points (x_i, x_j) corresponds to a segment such that points lying between x_i and x_j are approximated by the straight line joining $f(x_i)$ and $f(x_j)$. Given the desired number of segments H, and the metrics for the arc and path costs, our aim is to combine the n intervals into H segments by choosing a set of H+1 break points out of the n+1 break points such that the approximation error is minimized.

To do so, we define a network with n+1 nodes corresponding to interval limits $x_0, x_1, \ldots, x_n$ with an arc from node i to node j only if $i < j$, where $0 \le i, j \le n$. To find the *h-hop path* from node 0 to node j, we apply a recursive procedure that uses information about the (h-1) *hop path* to a node to determine the *h-hop path* to that node.

<u>Notation</u>

c_{ij}: cost of arc (i, j)

$L(j, h)$: length of shortest path from node 0 to node j containing h or fewer arcs where

$j = 1, 2, \ldots, n, h = 1, 2, \ldots, m$

Pred(j, h): predecessor of node j in the shortest h-hop path from node 1 to node j, where $j = 1, \ldots, n, h = 1, 2, \ldots, m$

To get the min-sum path, we use the basic recursion in Equation (4.1).

$$L(j,h) = Min\left(L(j,h-1), Min_{h-1 \le i \le j-1} L(i,h-1) + c_{ij}\right) \tag{4.1}$$

We observe that since arcs always go from lower-indexed nodes to higher-indexed nodes, node i should be considered a candidate intermediate node only if index i

is less than index *j*. Also, since the path from the source node to node *i* can have at most *i* arcs, we do not need to consider some node *i* if index *i* is less than *h*-1 since this option of coming to node *j* via node *i* would have been considered in a previous iteration of the algorithm.

The basic recursion for the min-max path follows the same idea and is given by Equation (4.2).

$$L(j,h) = Min\left(L(j,h-1),\ Min_{h-1 \le i \le j-1}\left(Max\left(L(i,h-1), c_{ij}\right)\right)\right). \tag{4.2}$$

We present the complete algorithm in Appendix A2. The computational complexity of the algorithm is O(n^2H) where the number of arcs in the graph is O(n^2).

One possible extension (which we have not implemented) is to use a two-phase approach that incorporates both the *min-sum* and the *min-max* metrics. In this approach, we first create and solve a shortest-path problem using the *min-max* metric, then create a new graph in which all the arcs with cost greater than the computed *min-max* cost are removed. We then solve a shortest path problem on this reduced graph using the *min-sum* metric. We note that in this approach, an arc can have two different arc lengths, one each for the two metrics.

4.2.2. Alternating Shortest Path-based Method

Since two-dimensional functions frequently arise in nonlinear programs, we now propose a heuristic method that uses the shortest path-based method to obtain good non-uniform Crisscross grids in two dimensions. We first observe that a Crisscross grid (Figure 4.5c) is a sequence of, what we call, vertical Crisscross patterns (Figure 4.5a) or horizontal Crisscross patterns (Figure 4.5b).

We cannot directly apply the one dimensional shortest path-based method to a two-dimensional function. Consider a two-dimensional function $f(x_1, x_2)$ such that $l_1 \leq x_1 \leq u_1$, $l_2 \leq x_2 \leq u_2$. Let us try to apply the shortest path-based method along the x_1-axis by dividing the $[l_1, u_1]$ range into small intervals and finding the best way of aggregating contiguous intervals into non-overlapping segments that cover the range $[l_1, u_1]$. However, as shown in Figure 4.6(a), each arc in the network over which we solve a shortest path problem now corresponds to a vertical pattern and to compute the arc cost, we need to know how the range $[l_2, u_2]$ has been divided into segments. Similarly, as shown in Figure 4.6(b), to apply the shortest path-based method along the x_2-axis and divide the range $[l_2, u_2]$ into non-overlapping segments, we need a network in which each arc corresponds to a horizontal pattern and to compute the arc cost, we need to know how the range $[l_1, u_1]$ has been divided into non-overlapping segments. Thus, it is not possible to decompose the two-dimensional problem into two one-dimensional problems.

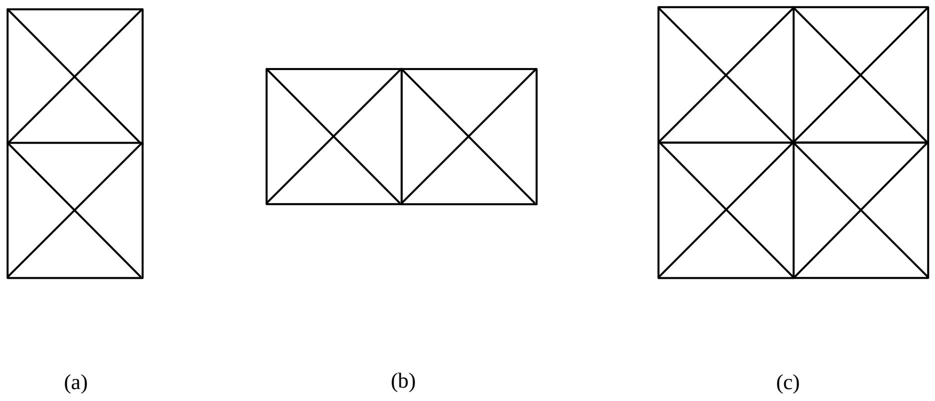
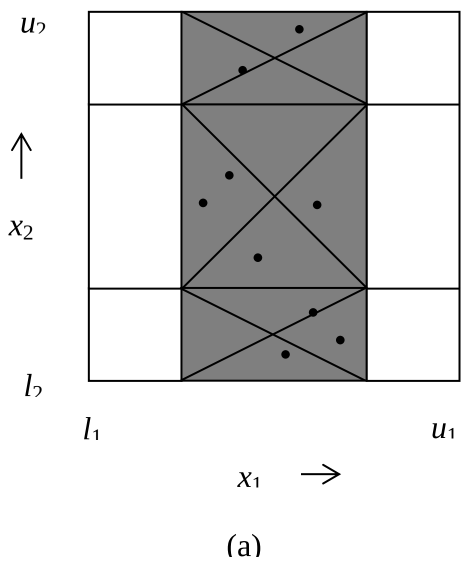


Figure 4.5: Building blocks of a Crisscross triangulation



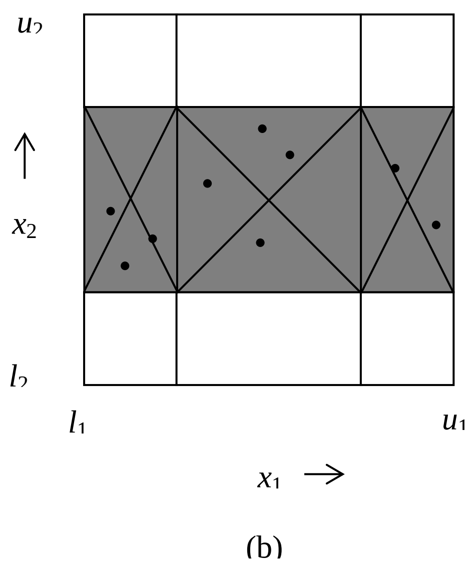


Figure 4.6: Computing approximation error for a Crisscross triangulation

To resolve this issue, we observe that if we fix the segments along one axis, then we can use the one-dimensional shortest path-based method to obtain a set of segments along the other axis. This suggests that we can apply an iterative procedure in which we start with an initial set of segments along an axis and solve a series of shortest path problems alternately along the x_1 and x_2 axes. Further, the shortest path-based segments computed for an axis at the end of an iteration are treated as fixed segments in the subsequent iteration. The iterative procedure stops when the application of the shortest path-based method along one axis is unable to improve the approximation quality of the grid. Since the number of possible combinations of intervals along the two axes is finite, the algorithm always terminates. Although the solution obtained by the algorithm is not necessarily optimal and depends upon the choice of the initial axis and the segments along that axis, the algorithm gave reasonably good solutions (compared to a uniform grid) for the test problems on which it was applied.

We provide the formal algorithm for generating non-uniform Crisscross grids using the alternating shortest path-based method in Appendix A3. A similar algorithm can be used to get a non-uniform Union Jack grid for a two-dimensional function. However, owing to the structure of the Union Jack triangulation, the network over which we solve a shortest path problem is more complex. We explain the method for getting good non-uniform Union Jack grids in Appendix A4.

Chapter 5: MIP Model Strengthening

For a given grid design, the effectiveness of the piecewise linear approximation approach depends upon how well we can solve the mixed integer program resulting from the reformulation and piecewise linearization of the nonlinear functions in the nonlinear program. In this chapter, we examine different techniques to strengthen the PLA-based mixed integer programming model. These techniques fall into three categories: (a) problem strengthening using shared grids across functions that have common variables, (b) problem strengthening using inequalities that arise when variables appearing in non-linear functions are also linked by constraints, and (c) problem strengthening using external bounds on variables. We discuss these three techniques in Sections 5.1, Section 5.2 and Section 5.3 respectively.

5.1 PROBLEM STRENGTHENING USING SHARED GRIDS

The traditional manner of solving nonlinear programs using the piecewise linear approximation approach entails creating individual mixed integer programming models for each of the nonlinear functions in the nonlinear program. We call such models *individual grid models*. However, when different nonlinear functions contain a common variable and the grids used for approximating these functions have the same granularity, then we can approximate these functions using a common grid and create for these functions a common mixed integer programming model that is smaller and tighter than the model with individual grids. We call such a model a *pattern-based model*. We can extend this concept further to handle situations in which different nonlinear functions with a common variable are approximated by grids of different granularities. In this case, we can conceptually create a single composite (or combined) grid that is an aggregation of the individual grids and define a single mixed integer programming model over this composite grid. This model, which we call a *combined partition model,* is even stronger than the pattern based models.

We divide Section 5.1 into three sections to discuss these three models in more detail. In Section 5.2.1, we discuss the individual grid models in which each function in the nonlinear program is linearized by an individual grid. Depending upon how adjacency is enforced, we can have different types of individual grid models. In the same way, different ways of ensuring adjacency lead to analgous types of pattern-based and combined partition models. We discuss the pattern-based models in Section 5.2.2 and the combined partition models in Section 5.2.3.

5.2.1 Individual Grid Models

The individual grid models are characterized by the presence of separate mixed integer programming models for each nonlinear function that appears in a nonlinear program. These models correspond to the traditional manner of applying piecewise linear approximation to solve nonlinear problems. We discuss these models here so that it is eaiser to understand the transition to the stonger pattern-based and combined partition models.

The models that we consider pertain to a single variable *X*. In an actual nonlinear program, the model will have to be defined for each variable that exists in a nonlinear function. Let *G* be the set of grids (functions) that contain variable *X* and *g* be a index over *P*. Let n_g be the number of segments in the partition for *X* in grid $g \in G$. Let I_g be the set of break points in the partition for *X* in grid $g \in G$ and *i* be an index over I_g such that I_g consists of $n_g + 1$ break points indexed from $i = 0$ to $i = n_g$ in the order of increasing *X*-values. Let x_i^g be the *X*-value corresponding to break point $i \in I_g$. Further, let us index the segments along the partition in grid $g \in G$ such that segment *i* corresponds to the interval between break points $i-1$ and *i*.

Let V^g be the set of vertices in grid $g \in G$ and $V^g(i)$ be the set of vertices in grid g that lie along break point $i \in I_g$. Let $\bar{x}_v^g$ and f_v be the *X*-value and the function value at vertex *v* respectively. Let $V^g(i)$ be the set of vertices in grid g that lie along break point $i \in I_g$. Let $\hat{h}^g$ be a variable that denotes the approximate function value for grid *g*. and α_v^g the vertex weight variable for vertex $v \in V^g$ in grid $g \in G$.

5.2.1.1 Individual Grid Model with Segment-wise Adjacency

In an individual grid model with segmentwise adjacency, we define two additional sets of variables.

- Z_i^g : binary variables defined for each segment $i \in I_g$ in grid $g \in G$ such that Z_i^g is one if we select segment $i \in I_g$ in grid $g \in G$, and is zero otherwise
- λ_i^g : marginal weight variable for break point $i \in I_g$ in grid $g \in G$ representing the sum of the vertex weight variables along break point *i*

Given this setting, we can define a mixed integer programming model that selects in each grid, a single segment along the *X* partition such that only the vertices that lie along the selected segment can have a positive weight and the vertex weight variables sum to one.

1) Function value constraint

The approximate function value for each grid is a convex combination of the function values at the vertices of that grid.

$$\hat{h}^g = \sum_{v \in V^g} \alpha_v^g f_v \qquad \text{for all } g \in G \tag{5.1}$$

2) Marginal weight constraint

For each break point $i \in I_g$ in grid $g \in G$, the marginal weight is the sum of the vertex weight variables along that break point.

$$\lambda_i^g = \sum_{v \in V^g(i)} \alpha_v^g \qquad \text{for all } g \in G, \text{ for all } i = 0,\dots,n_g \tag{5.2}$$

3) Variable value constraint

In each grid, the value that a variable takes is a convex combination of the variable values at the vertices of that grid. The variable value can also be expressed in terms of the marginal weight variables as shown in Equation (5.4).

$$X = \sum_{v \in V^g} \alpha_v^g \bar{x}_v^g \qquad \text{for all } g \in G \tag{5.3}$$

$$X = \sum_{i=0}^{n_g} \sum_{v \in V^g(i)} \alpha_v^g \bar{x}_v^g \qquad \text{for all } g \in G$$

$$X = \sum_{i=0}^{n_g} x_i^g \left(\sum_{v \in V^g(i)} \alpha_v^g \right) \qquad \text{for all } g \in G$$

$$X = \sum_{i=0}^{n_g} x_i^g \lambda_i^g \qquad \text{for all } g \in G \tag{5.4}$$

4) Convex combination constraint

In each grid, the sum of the vertex weight variables is equal to one. Equivalently, we could say that the marginal weight variables sum to one.

$$\sum_{v \in V^g} \alpha_v^g = 1 \qquad \text{for all } g \in G \tag{5.5}$$

$$\sum_{i=0}^{n_g} \lambda_i^g = 1 \text{ for all } g \in G \tag{5.6}$$

5) Segment-wise adjacency conditions

These constraints ensure that for each grid, a single segment is selected from the *X* partition and only those marginal weight variables that correspond to the end points of the selected segment can be strictly positive, which implies that only those vertices that lie along the end points of the selected segment can have a positive vertex weight.

$$\sum_{i=1}^{n_g} Z_i^g = 1 \quad \text{for all } g \in G \tag{5.7}$$

$$\lambda_{i-1}^g + \lambda_i^g \geq Z_i^g \quad \text{for all } g \in G, \text{ for all } i \in \{1,\ldots,n_g\} \tag{5.8}$$

$$\lambda_0^g \leq Z_1^g \quad \text{for all } g \in G \tag{5.9a}$$

$$\lambda_i^g \leq Z_i^g + Z_{i+1}^g \quad \text{for all } g \in G, i \in \{1,\ldots,n_g - 1\} \tag{5.9b}$$

$$\lambda_{n_g}^g \leq Z_{n_g}^g \quad \text{for all } g \in G \tag{5.9c}$$

6) Non-negativity constraints

The vertex weight variables must be non-negative, which also ensures that the marginal weight variables are non-negative.

$$\alpha_v^g \geq 0 \quad \text{for all } g \in G, v \in V^g \tag{5.10}$$

7) Binary variable constraints

$$Z_i^g \in \{0,1\} \text{ for all } i \in \{1,\ldots n_g\} \tag{5.11}$$

We call the model formed by Equations (5.1–5.11) an *individual grid model with segment-wise adjacency* (IND-SEG) because the adjacency conditions are defined for each individual segment. In the next section, we show how we can define adjacency conditions for groups of segments and create a stronger individual grid model.

5.2.1.2 Individual Grid Model with Cumulative Adjacency

The model defined by Equations (5.1–5.11) is not locally ideal i.e., the extreme points of the polyhedron describing the piecewise linear function of *X*, without additional constraints may have fractional extreme points. However, Padberg (2000) shows that by defining the adjacency conditions over groups of segments, we can make the model locally ideal. This entails replacing the segment-wise adjacency constraints (Equations 5.8–5.9) with the following constraints.

$$\sum_{i'=i}^{n_g} \lambda_{i'}^{g} \le \sum_{i'=i}^{n_g} Z_{i'}^{g} \quad \text{for all } \; g \in G, \text{ for all } i = 2,\dots,n_g \tag{5.12a}$$

$$\sum_{i'=i}^{n_g} \lambda_{i'}^{g} \ge \sum_{i'=i+1}^{n_g} Z_{i'}^{g} \quad \text{for all } \; g \in G, \text{ for all } i = 1,\dots,n_g - 1 \tag{5.12b}$$

We now propose a different way of writing Equation (5.12) by creating auxiliary binary variables that represent the cumulative segment selection decision. Specifically, for each segment in the partition for *X* in grid $g \in G$, let W_i^g denote the cumulative segment selection (binary) variable that takes the value one if the model selects segment *i* or higher, defined for *i* = 1, …, n_g.

$$W_i^g = \sum_{i'=i}^{n_g} Z_{i'}^{g} \quad \text{for all } i = 1,\dots,n_g$$

Similarly, we create cumulative marginal weight variables for each break point in each pattern defined as follows.

$$\omega_i^g = \sum_{i'=i}^{n_g} \lambda_{i'}^{g} \quad \text{for all } i = 0,\dots,n_g$$

Using these cumulative variables, we create the *cumulative adjacency constraints* as shown in Equations (5.13–5.15). These constraints are less dense than the original constraints (Equation 5.12) and can possibly reduce the chances of Cplex eliminating the variables in these consrtaints during preprocessing. Note that we need Equation 5.13(c) since $W_i^g = Z_i^g + W_{i+1}^g$ for all $i = 1,...,n_g - 1$.

$$\omega_i^g \le W_i^g \quad \text{for all } g \in G, \text{ for all } i = 2,...,n_g \tag{5.13a}$$

$$\omega_i^g \ge W_{i+1}^g \quad \text{for all } g \in G, \text{ for all } i = 1,...,n_g - 1 \tag{5.13b}$$

$$W_i^g \ge W_{i+1}^g \quad \text{for all } g \in G, \text{ for all } i = 1,...,n_g - 1 \tag{5.13c}$$

$$\omega_0^g = 1 \quad \text{for all } g \in G \tag{5.14a}$$

$$W_1^g = 1 \quad \text{for all } g \in G \tag{5.14b}$$

$$W_i^g \in \{0,1\} \quad \text{for all } g \in G, \text{ for all } i = 1,...,n_g \tag{5.15}$$

Further, we replace the *variable value constraint* (Equation 5.4) with the *cumulative variable value constraint* (Equation 5.16), and the *marginal weight constraint* (Equation 5.2) with the *cumulative marginal weight constraint* (Equations 5.17).

$$X = \sum_{i=0}^{n_g - 1} x_i(\omega_i^g - \omega_{i+1}^g) + x_{n_g}\omega_{n_g}^g \quad \text{for all } g \in G \tag{5.16}$$

$$\sum_{v \in V^g(i)} \alpha_v^g = \omega_i^g - \omega_{i+1}^g \quad \text{for all } g \in G, \text{ for all } i = 0,...,n_g - 1 \tag{5.17a}$$

$$\sum_{v \in V^g(i)} \alpha_v^g = \omega_i^g \quad \text{for all } g \in G, \text{ for all } i = n_g \tag{5.17b}$$

We call the model formed by the *function value constraint* (Equation 5.1), non-negativity constraint (Equation 5.10), and Equations (5.13–5.17) an *individual grid model with cumulative adjacency* (IND-CUM) because the adjacency conditions are defined cumulatively and not for individual segments. As shown in Appendix A11, the cumulative adjacency model has the same LP-relaxation value as the segment-wise adjacency model. However, since the cumulative adjacency model is locally ideal, it requires fewer branch and bound nodes and therefore can be solved in less time than that required for the segment-wise adjacency model.

5.2.1.3 Individual Grid Model with Logarithmic Indexing

As discussed earlier, Vielma and Nemhauser (2011) show that we do not need *n* binary variables for enforcing adjacency across *n* segments. Instead, we can do so by using $log_2(n)$ binary variables such that each segment is assigned a binary address that corresponds to a reflected binary code or gray code. A gray code is defined as a logarithmic binary indexing scheme in which two successive values differ in only one bit (Gilbert 1957). For example, intervals 0, 1, 2, and 3 can be identified by using two bits by assigning the reflected binary codes 00, 01, 11 and 10. Similarly, intervals 0, 1, 2, 3, 4, 5, 6, and 7 can be identified by using three bits by assigning the reflected binary codes 000, 001, 011, 010, 110, 111, 101 and 100.

Given that we have n_g segments in the partition for *X* in grid $g \in G$, let us assign a reflected binary code comprised of $\text{ceil}(\log_2(n_g))$ bits to each of these segments. Let $\bar{I}_g(q)$ be the set of break points in the partition for *X* in grid $g \in G$ such that each break point serves as the end point of at least one segment whose reflected binary code has a zero at position *q*. Similarly, let $\hat{I}_g(q)$ be the set of break points in the partition for *X* in grid $g \in G$ such that each break point serves as the end point of at least one segment whose reflected binary code has a one at position *q*.

Let us define binary variables $B_q^g, q = 1, ..., \log_2(n_g)$ such that B_q^g takes the value one if we select in grid $g \in G$ an *X*-segment whose reflected binary code contains a one at position *q* and B_q^g takes the value zero if we select in grid $g \in G$ an *X*-segment whose reflected binary code contains a zero at position *q*. Using these variables, we define the *logarithmic adjacency constraints* as per Equations (5.18–5.19).

$$\sum_{i \notin \bar{I}_g(q)} \lambda_i^g \le B_q^g \text{ for all } q = 1,\ 2, \ldots, \text{ceil}[\log_2(n_g)], \text{ for all } g \in G \tag{5.18a}$$

$$\sum_{i \notin \hat{I}_g(q)} \lambda_i^g \le 1 - B_q^g \text{ for all } q = 1, 2, \ldots, \text{ceil}[\log_2(n_g)], \text{ for all } g \in G \tag{5.18b}$$

$$B_q^g \in \{0,1\} \text{ for all } q = 1, 2, \ldots, \text{ceil}[\log_2(n_g)], \text{ for all } g \in G \tag{5.19}$$

Equation (5.18a) ensures that if we select in grid $g \in G$ an *X*-segment whose reflected binary code has a zero at position *q* (i.e., if B_q^g is zero), then all the break points that do not belong to $\bar{I}_g(q)$ are assigned a marginal weight of zero, which also ensures that the vertices corresponding to those break points are also assigned a vertex weight of zero. On the other hand, Equation (5.18b) ensures that if we select in grid $g \in G$ an *X*-segment whose reflected binary code has a one at position *q* (i.e., if B_q^g is one), then all the break points that do not belong to $\hat{I}_g(q)$ are assigned a marginal weight of zero.

We call the model formed by the *function value constraint* (Equations 5.1), *marginal weight constraint* (Equation 5.2), *variable value constraint* (Equation 5.4), *convex combination constraint* (Equation 5.6), *non-negativity constraint* (Equation 5.10), and *logarithmic adjacency constraints* (Equations 5.18–5.19) an *individual grid model with logarithmic adjacency* (IND-LOG).

5.2.1.4 Individual Grid Model with SOS-2-based adjacency

We can also ensure adjacency by declaring the set of marginal weight variables in each grid a Special Ordered Set of Type 2.

$$\lambda_0^g, \ldots, \lambda_{n_g}^g \text{ SOS} - 2 \text{ for all } g \in G \tag{5.20}$$

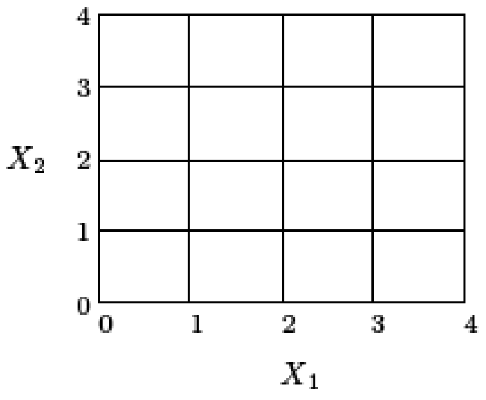
We call the model formed by the *function value constraint* (Equations 5.1), *marginal weight constraint* (Equation 5.2), *variable value constraint* (Equation 5.4), *convex combination constraint* (Equation 5.6), *non-negativity constraint* (Equation 5.10), and Equations (5.20) an *individual grid model with SOS-II constraints* (IND-SOS). The notation used in the different individual grid models in Appenix A12.

In the next section, we discuss the pattern-based models that use common grids for functions that (a) share a common variable, and (b) are linearized using grids of the same granularity.

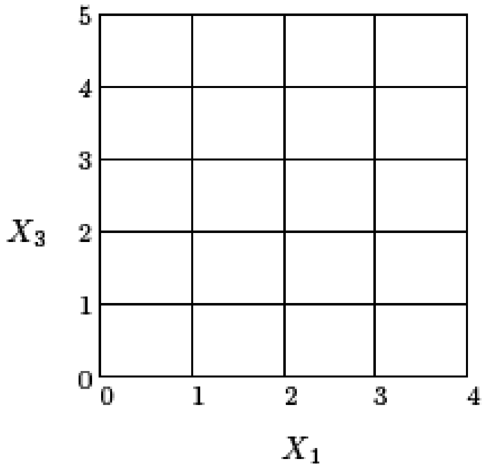
5.2.2 Pattern-based Models

Before discussing the formulation for the pattern-based models, we first consider an example that conveys the intuition behind these models. Let us assume that we have a nonlinear program that is linear except for the presence of two nonlinear functions involving three bounded variables X_1, X_2, and X_3 such that $f_1(X_1, X_2)$ is a non-separable function of X_1 and X_2, and $f_2(X_1, X_3)$ is a non-separable function of X_1 and X_3. To apply the PLA approach to this problem, we

a) partition the domain of f_1 with a rectangular 4×4 (i.e., 4 segment $\times$ 4 segment) uniform grid G_1,
b) partition the domain of f_2 with a rectangular 4×5 grid G_2, and
c) create a mixed integer program that ensures that in each grid, (i) we select only one cell, and (ii) the approximate function value is given by a convex combination of the function values at the vertices of the selected cell.



Grid G$_1$



Grid G$_2$

Figure 5.1: Domain partitioning for functions f_1 and f_2

To create a mixed integer program, we index the break points for variables X_1, X_2, and X_3 using indices *i*, *j* and *k* where $0 \le i \le 4$, $0 \le j \le 4$ and $0 \le k \le 5$. Let x_1^i, x_2^j, and x_3^k be the values of X_1, X_2, and X_3 at break points *i*, *j* and *k* respectively. Let us define vertex weight variables α_{ij}^1 and α_{ik}^2 for grids G_1 and G_2 respectively as continuous variables bounded between zero and one. Let $\hat{h}_1$ and $\hat{h}_2$ be the approximate function values of f_1 and f_2 respectively. The segment-wise adjacency model for these two grids is comprised of the following constraints.

1) Function value constraint

The approximate function value for each grid is a convex combination of the function values at the vertices of that grid.

$$\hat{h}_1 = \sum_{i=0}^{4}\sum_{j=0}^{4}\alpha_{ij}^1 f(x_1^i, x_2^j) \qquad \hat{h}_2 = \sum_{i=0}^{4}\sum_{k=0}^{5}\alpha_{ik}^2 f(x_1^i, x_3^k) \qquad (5.21)$$

2) Variable value constraint

$$X_1 = \sum_{i=0}^{4}\sum_{j=0}^{4}\alpha_{ij}^1 x_1^i \qquad X_2 = \sum_{i=0}^{4}\sum_{j=0}^{4}\alpha_{ij}^1 x_2^j \tag{5.22a}$$

$$X_1 = \sum_{i=0}^{4}\sum_{k=0}^{5}\alpha_{ik}^2 x_1^i \qquad X_3 = \sum_{i=0}^{4}\sum_{k=0}^{5}\alpha_{ik}^2 x_3^k \tag{5.22b}$$

In terms of the marginal wights, Equation (5.22a) can be expressed using Equation (5.23).

$$X_1 = \sum_{i=0}^{4}\left(x_1^i \sum_{j=0}^{4}\alpha_{ij}^1 \right) \quad \text{or} \quad X_1 = \sum_{i=0}^{4}\left(x_1^i \lambda_{1i}^1 \right) \quad \text{where} \quad \lambda_{1i}^1 = \sum_{j=0}^{4}\alpha_{ij}^1 \tag{5.23a}$$

$$X_2 = \sum_{j=0}^{4}\left(x_2^j \sum_{i=0}^{4}\alpha_{ij}^1 \right) \quad \text{or} \quad X_2 = \sum_{j=0}^{4}\left(x_2^j \lambda_{2j}^1 \right) \quad \text{where} \quad \lambda_{2j}^1 = \sum_{i=0}^{4}\alpha_{ij}^1 \tag{5.23b}$$

We call λ_{1i}^1 the *marginal weight* associated with break point *i* for variable X_1 in grid G_1 and it is equal to the sum of the vertex weights across all vertices in G_1 that lie along break point *i*. Similarly λ_{2j}^1 is the marginal weight associated with break point *j* for variable X_2 in grid G_1.

Similarly, we can express Equation (5.22b) in terms of the marginal weights as Equation (5.24).

$$X_1 = \sum_{i=0}^{4}\left(x_1^i \sum_{k=0}^{5}\alpha_{ik}^2 \right) \quad \text{or} \quad X_1 = \sum_{i=0}^{4}\left(x_1^i \lambda_{1i}^2 \right) \quad \text{where} \quad \lambda_{1i}^2 = \sum_{k=0}^{5}\alpha_{ik}^2 \tag{5.24a}$$

$$X_3 = \sum_{k=0}^{5}\left(x_3^k \sum_{i=0}^{4}\alpha_{ik}^2 \right) \quad \text{or} \quad X_3 = \sum_{k=0}^{5}\left(x_3^k \lambda_{3k}^2 \right) \quad \text{where} \quad \lambda_{3k}^2 = \sum_{i=0}^{4}\alpha_{ik}^2 \tag{5.24b}$$

Here λ_{1i}^2 is the marginal weight associated with break point *i* for variable X_1 in grid G_2 and λ_{3k}^2 is the marginal weight associated with break point *k* for variable X_3 in grid G_2.

3) Convex combination constraint

These constraints ensure that the sum of the vertex weights is equal to one.

$$\sum_{i=0}^{4}\sum_{j=0}^{4}\alpha_{ij}^1 = 1 \tag{5.25a}$$

$$\sum_{i=0}^{4}\sum_{k=0}^{5}\alpha_{ik}^{2}=1 \tag{5.25b}$$

4) Adjacency constraints

We need to ensure that exactly one cell is chosen in each grid. This can be done by ensuring that exactly one segment is chosen along each dimension in a grid and the marginal weight variables at the two ends of the chosen segment sum to one. Let Z_{1i}^{1} be a binary variable that takes value one if we select segment *i* for variable X_1 in grid G_1, and is zero otherwise. Let Z_{2j}^{1} be a binary variable that takes value one if we select segment *j* for variable X_2 in grid G_1, and is zero otherwise. Let Z_{1i}^{2} be a binary variable that takes value one if we select segment *i* for variable X_1 in grid G_2, and is zero otherwise. Let Z_{3k}^{2} be a binary variable that takes value one if we select segment *k* for variable X_3 in grid G_2, and is zero otherwise. Then, the adjacency conditions for Grid G_1 and G_2 are given by Equations (5.26) and Equations (5.27) respectively.

$$\sum_{i=1}^{4} Z_{1i}^{1}=1,\ \lambda_{1i}^{1}+\lambda_{1,i-1}^{1}\ge Z_{1i}^{1} \quad \text{for all } i\in\{1,...,4\} \tag{5.26a}$$

$$\sum_{j=1}^{4} Z_{2j}^{1}=1,\ \lambda_{2j}^{1}+\lambda_{2,j-1}^{1}\ge Z_{2j}^{1} \quad \text{for all } j\in\{1,...,4\} \tag{5.26b}$$

$$\sum_{i=1}^{4} Z_{1i}^{2}=1,\ \lambda_{1i}^{2}+\lambda_{1,i-1}^{2}\ge Z_{1i}^{2} \quad \text{for all } i\in\{1,...,4\} \tag{5.27a}$$

$$\sum_{k=1}^{5} Z_{3k}^{2}=1,\ \lambda_{3k}^{2}+\lambda_{3,k-1}^{2}\ge Z_{3k}^{2} \quad \text{for all } k\in\{1,...,5\} \tag{5.27b}$$

5) Non-negativity and binary variable constraints

$$\alpha_{ij}^{1}\ge 0 \text{ for all } i\in\{0,...,4\},\ j\in\{0,...,4\} \tag{5.28a}$$

$$\alpha_{ik}^{2}\ge 0 \text{ for all } i\in\{0,...,4\},\ k\in\{0,...,5\} \tag{5.28b}$$

$$Z_{1i}^{1}\in\{0,1\} \text{ for all } i\in\{1,...,4\} \tag{5.28c}$$

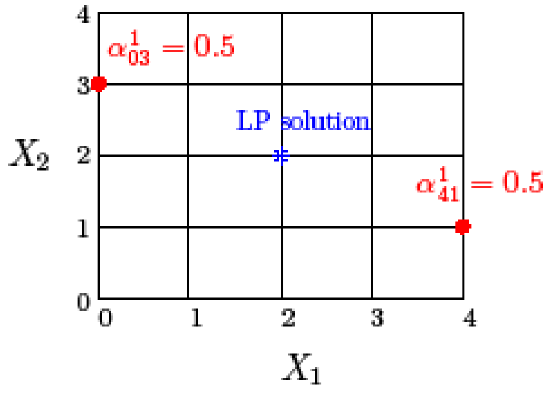
$$Z_{2j}^{1} \in \{0,1\} \text{ for all } j \in \{1,\ldots,4\} \tag{5.28d}$$

$$Z_{1i}^{2} \in \{0,1\} \text{ for all } i \in \{1,\ldots,4\} \tag{5.28e}$$

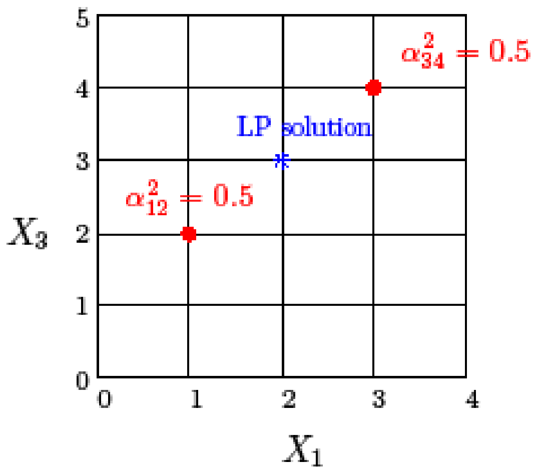
$$Z_{3k}^{2} \in \{0,1\} \text{ for all } k \in \{1,\ldots,5\} \tag{5.28f}$$

The key observation is that although X_1 appears in two different functions, yet it will have a single value in any solution to the mixed integer program. Further, the value of X_1 will belong to the same segment in both the grids and will also have the same set of marginal weights in both the grids. In other words, λ_{1i}^{1} will be equal to λ_{1i}^{2} for all *i*. Thus, we can define a single set of marginal weights for X_1 and thereby create a smaller and tighter model. We call such a model a *patter-based model* where a pattern is a partition induced by a set of break points along the domain of a function corresponding to a specific variable.

A pattern-based model is not only smaller but also tighter than a traditional grid-based model, which implies that there are fractional LP solutions that are feasible for the grid-based model but not for the pattern-based model. Consider the example shown in Figure 5.2 where we have an LP solution that is feasible for a grid-based model. However, this LP solution is not feasible for the pattern-based model since (a) the marginal weight corresponding to break points 0 and 4 (along the *X*-axis) have a value of 0.5 each in Grid G_1 but values of zero each in Grid G_2, and (b) the marginal weight corresponding to break points 1 and 3 (along the *X*-axis) have a value of 0 each in Grid G_1 but values of 0.5 each in Grid G_2. Thus, the LP relaxation value of a pattern-based model can be higher than that of a grid-based model, which implies that the MIP solver might find it easier to solve a pattern-based model.



Grid G_1



Grid G_2

Figure 5.2: LP solution that is not feasible for a *pattern-based* model

Just as we had four types of grid-based models, we can have four types of pattern based models: pattern-based models with segment-wise adjacency, pattern-based models with cumulative adjacency, pattern-based models with logarithmic adjacency, and pattern-based models with SOS-2 based adjacency. We now describe these models in more detail.

We first define some additional notation. For a comprehensive list of all the symbols, refer to Appendix A12. Let G be the set of grids (functions) that contain variable X and let P be the set of partitioning patterns for variable X used in the linear approximations of all functions that contain variable X. Let g and p be indices over G and P respectively. In the example discussed earlier, G will contain grids G_1 and G_2.

Let n_p be the number of break points in partitioning pattern $p \in P$ and J_p be the set of break points in p such that J_p consists of n_p break points indexed from j = 0 to j = n_p in the order of increasing X-values. In the example discussed earlier, there is a single partition (with 5 break points) for variable X.

Let G_p be the set of grids (functions) that contain variable X and use pattern $p \in P$. Let $p(g)$ be the pattern used for X in grid g. Let V^g be the set of vertices in grid $g \in G$ and $V^g(j)$ be the set of vertices in grid g that lie along break point $j \in J_p$. Let $\hat{h}^g$ be the approximate function value for grid g. Let x_j^p be the X-value corresponding to break point $j \in J_p$ in pattern $p \in P$.

Using the notation defined above, we now discuss the four types of pattern-based models.

5.2.2.1 Pattern-based Model with Segment-wise Adjacency

To create the segment-wise adjacency constraints, we define two additional types of variables. Let σ_j^p denote the (continuous) marginal weight variables defined for each break point $j \in \{0,...,n_p\}$ in each pattern $p \in P$, and S_j^p be the binary variables defined for each segment $j \in \{1,...,n_p\}$ in each pattern $p \in P$.

1) Pattern-based variable value constraint

In each pattern, the value taken by X is a convex combination of the X-values corresponding to the break points of that pattern.

$$X = \sum_{j=0}^{n_p} x_j^p \sigma_j^p \quad \text{for all} \quad p \in P \tag{5.29}$$

2) Pattern-based convex combination constraint

For each pattern, the marginal weight variables sum to one.

$$\sum_{j=0}^{n_p} \sigma_j^p = 1 \quad \text{for all } p \in P \tag{5.30}$$

3) Pattern-based marginal weight constraint

For each grid, the sum of the vertex weight variables corresponding to a specific break point is equal to the marginal weight variable corresponding to that break point in the specific pattern (for *X*) that we use in that grid.

$$\sum_{v \in V^g(j)} \alpha_v^g = \sigma_j^{p(g)} \quad \text{for all } g \in G, j = 0, \ldots, n_{p(g)} \tag{5.31}$$

4) Segment-wise adjacency constraints

These constraints ensure that (a) in each pattern for *X*, a single segment is selected, and (b) the marginal weight variables corresponding to the end points of only the selected segment can be strictly positive.

$$\sum_{j=1}^{n_p} S_j^p = 1 \quad \text{for all} \quad p \in P \tag{5.32}$$

$$\sigma_{j-1}^p + \sigma_j^p \geq S_j^p \quad \text{for all } p \in P, j \in \{1, \ldots, n_p\} \tag{5.33}$$

$$\sigma_0^p \leq S_1^p \quad \text{for all } p \in P \tag{5.34a}$$

$$\sigma_j^p \leq S_j^p + S_{j+1}^p \quad \text{for all } p \in P, j \in \{1, \ldots, n_p - 1\} \tag{5.34b}$$

$$\sigma_{n_p}^p \leq S_{n_p}^p \quad \text{for all } p \in P \tag{5.34c}$$

5) Binary variable constraints

$$S_j^p \in \{0,1\} \quad \text{for all } p \in P, \text{ for all } j = 1, \ldots, n_p \tag{5.35}$$

We call the model formed by the *function value constraint* (Equation 5.1), and Equations (5.29–5.35) a *pattern-based model with segment-wise adjacency* (PAT-SEG).

5.2.2.2 Pattern-based Model with Cumulative Adjacency

As in the individual grid model, we can create a pattern-based model using cumulative adjacency conditions. To do so, we define for each partition *p* of the variable *X*, a variable T_j^p that denotes the cumulative segment selection (binary) variable that takes the value one if the model selects segment *j* or higher, defined for *j* = 1, …, n_p. Similarly, we define τ_j^p as the cumulative marginal weight variable for each break point in each pattern.

$$T_j^p = \sum_{j'=j}^{n_p} S_{j'}^p \text{ for all } j = 1,\dots,n_p$$

$$\tau_j^p = \sum_{j'=j}^{n_p} \sigma_{j'}^p \text{ for all } j = 0,\dots,n_p$$

Using these cumulative variables, we create the *pattern-based cumulative adjacency constraint* as shown in Equation (5.36).

$$\tau_j^p \le T_j^p \quad \text{for all } j = 2,\dots,n_p,\ p \in P \tag{5.36a}$$

$$\tau_j^p \ge T_{j+1}^p \quad \text{for all } j = 1,\dots,n_p - 1,\ p \in P \tag{5.36b}$$

$$\tau_0^p = 1 \quad \text{for all } p \in P \tag{5.36c}$$

$$T_1^p = 1 \quad \text{for all } p \in P \tag{5.36d}$$

$$T_j^p \ge T_{j+1}^p \quad \text{for all } j = 1,\dots,n_p - 1,\ p \in P \tag{5.36e}$$

We also need the *pattern-based cumulative variable value constraint* (Equation 5.37), and the *pattern-based cumulative marginal weight constraint* (Equation 5.38).

$$X = \sum_{j=0}^{n_p - 1} x_j^p(\tau_j^p - \tau_{j+1}^p) + x_{n_p}^p \tau_{n_p}^p \quad \text{for all } p \in P \tag{5.37}$$

$$\sum_{v \in V^g(j)} \alpha_v^g = \tau_j^p - \tau_{j+1}^p \quad \text{for all } g \in G, j = 0, ..., n_{p(g)} - 1 \tag{5.38a}$$

$$\sum_{v \in V^g(j)} \alpha_v^g = \tau_j^p \quad \text{for all } g \in G, \ j = n_{p(g)} \tag{5.38b}$$

We call the model formed by the *function value constraint* (Equation 5.1), non-negativity constraint (Equation 5.10), and Equations (5.36–5.38) a *pattern-based model with cumulative adjacency* (PAT-CUM).

5.2.2.3 Pattern-based Model with Logarithmic Indexing

As in the individual grid model, we can have a log-based pattern-based model. Given that we have n_p segments in partition *p*, let us assign a reflected binary code comprised of $\text{ceil}(\log_2(n_p))$ bits to each of these segments. Let $\bar{J}_p(q)$ be the set of break points in pattern *p* such that each break point serves as the vertex of at least one cell whose reflected binary code contains a zero at the position *q*. Similarly, let $\hat{J}_p(q)$ be the set of break points in pattern *p* such that each break point serves as the vertex of at least one cell whose reflected binary code contains a one at position *q*.

Let us define binary variables $C_q^p, \text{for } q = 1, ..., \log_2(n_p), \ p \in P$ such that C_q^p takes the value one if we select a cell whose reflected binary code contains a one at position *q* and C_q^p takes the value zero if we select a cell whose reflected binary code contains a zero at position *q*. We then have the following adjacency constraints.

.

$$\sum_{j \notin \bar{J}_p(q)} \sigma_j^p \le C_q^p \ \text{ for all } q = 1, 2, ..., \text{ceil}[\log_2(n_p)] \text{ for all } \ p \in P \tag{5.39a}$$

$$\sum_{j \notin \hat{J}_p(q)} \sigma_j^p \le 1 - C_q^p \ \text{ for all } q = 1, 2, ..., \text{ceil}[\log_2(n_p)], \text{ for all } \ p \in P \tag{5.39b}$$

$$C_q^p \in \{0,1\} \text{ for all } q = 1, 2, ..., \text{ceil}[\log_2(n_p)], \text{ for all } \ p \in P \tag{5.40}$$

Equation (5.39a) ensures that if we select a segment in pattern *p* whose reflected binary code has a zero at position *q* (i.e., if C_q^p is zero), then all the break points that do not belong to $\bar{J}_p(q)$ are assigned a marginal weight of zero, which also ensures that the vertices corresponding to that break point are also assigned a vertex weight of zero. On the other hand, Equation (5.39b) ensures that if we select a segment in pattern *p* whose reflected binary value has a one at position *q* (i.e., if C_q^p is one), then all the break points that do not belong to $\hat{J}_p(q)$ are assigned a marginal weight of zero.

We call the model formed by the *function value constraint* (Equations 5.1), *non-negativity constraint* (Equation 5.10), pattern-based variable value constraint (Equation 5.29), pattern-based convex combination constraint (Equation 5.30), pattern-based marginal weight constraint (Equation 5.31), and Equations (5.39–5.40) a *pattern-based model with logarithmic adjacency* (PAT-LOG).

5.2.2.4 Pattern-based model with SOS-2-based adjacency

As in the individual grid models, adjacency can also be enforced by defining for each pattern, the set of marginal weight variables as a Special Ordered Set of Type 2.

$$\sigma_0^p,\ldots,\sigma_{n_p}^p \ \text{SOS}-2 \ \text{for all} \quad p \in P \tag{5.41}$$

We call the model formed by the *function value constraint* (Equations 5.1), *non-negativity constraint* (Equation 5.10), pattern-based variable value constraint (Equation 5.29), pattern-based convex combination constraint (Equation 5.30), pattern-based marginal weight constraint (Equation 5.31), and Equation (5.41) a *pattern-based model* *with SOS-2 adjacency* (PAT-SOS). The notation used in the different pattern-based models is summarized in Appenix A12.

5.2.3 Combined Partition Models

The pattern-based models discussed in the previous section take advantage of common partitions of the same variable in different grids. However, in a pattern–based model, the adjacency conditions have to be defined for each pattern. If we have multiple patterns for a given variable, then we could further a stronger model by defining a combined partition that contains all the break points that occur in the various partitions for a given variable. We then define a single set of adjacency conditions for this combined partition and express the marginal weights in the individual patterns in terms of the marginal weights in the combined partition. We show a combined partition in Figure 5.3. As with pattern-based models, we can have four types of combined partition models: combined partition models with segment-wise adjacency, combined partition models with cumulative adjacency, combined partition models with logarithmic adjacency, and combined partition models with SOS-2 based adjacency. We now discuss these models in more detail.

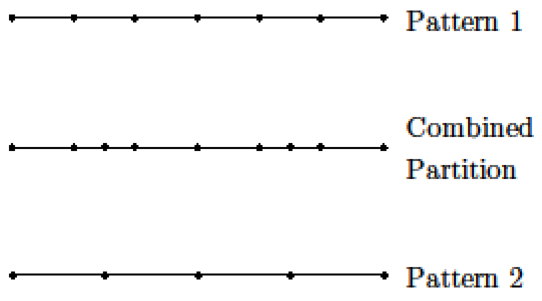


Figure 5.3: Example of a combined partition

We recall that n_p denotes the number of break points in partitioning pattern $p \in P$ and J_p is the set of break points in p such that J_p consists of n_p break points indexed from *j* = 0 to *j* = n_p in the order of increasing *X*-values. Let the combined partition contain *m* break points and let *K* denote the set of these break points. Let *k* be an index over the break point (and segments) in the combined partition such that *k* can take values over the set { $0,\ldots,m$ }. Let x_k , $k \in K$ be the value of *X* at break point *k* in the combined partition and let x_j^p be the value of *X* at break point *j* in pattern *p*. Let $j(k, p)$ be the closest break point in pattern *p* that either coincides with break point *k* of the combined partition or lies strictly to the right of break point *k* of the combined partition, i.e., $j(k, p) = Min\ j : x_j^p \geq x_k$, for all $p \in P, k \in K$.

Let $k(j, p)$ be the break point in the combined partition that coincides with break point *i* in pattern *p*, i.e. , $k(j, p) = k : x_j^p = x_k$, for all $p \in P$, for all $j \in J_p$.

Let $K_p(j)$ be the set of break points in the combined partition that lie in the interior of segment *j* in partition *p*, defined for all $j \in J_p \setminus \{0\}, p \in P$. In other words, $K_p(j) = \{k(j-1, p)+1, \ldots k(j, p)-1\}$. Further, let us define a parameter θ_{kp} for all $k = \{0, \ldots, m\}$ and $p \in P$ as follows.

$$\theta_{kp} = \frac{x_k - x^p_{j(k,p)-1}}{x_{j(k,p)} - x^p_{j(k,p)-1}} \text{ for all } p \in P, k \in K \setminus \{0\}. \tag{5.42}$$

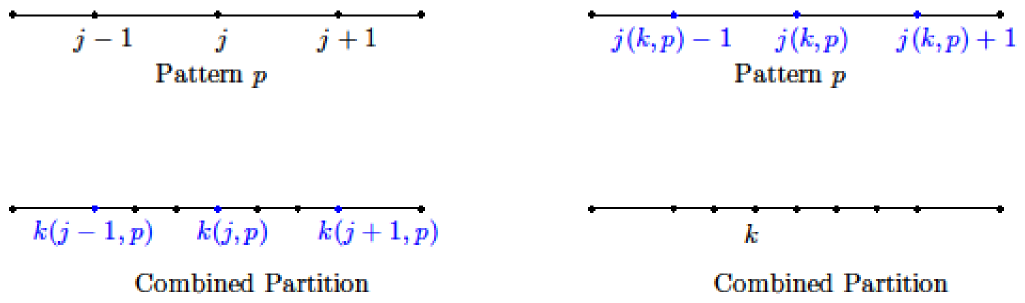


Figure 5.4: Relating break point indices in individual patterns and the combined partition

5.2.3.1 Combined Partition Model with Segment-wise Adjacency

Let ρ_k be the *marginal weight variable* for breakpoint *k* in the combined partition, defined for $k = \{0, \ldots, m\}$. We enforce adjacency by defining binary variables for each segment in the combined partition and then adding forcing constraints that ensure that in the combined partition (a) exactly one segment is chosen, and (b) only the marginal weight variables corresponding to the end points of the chosen segment can have positive weights. Let us define a binary variable $R_k, k \in \{1, \ldots m\}$ that is one if we select segment *k* in the combined partition.

1) Variable value constraint for combined partition

The variable value constraint, defined for the combined partition, ensures that the value of the variable (*X*) is equal to a convex combination of the variable values corresponding to the break points in the combined partition.

$$X = \sum_{k=0}^{m} \rho_k x_k \tag{5.43}$$

2) Convex combination constraint for combined partition

The sum of the marginal weights across all the break points of the combined partition must sum to one.

$$\sum_{k=0}^{m} \rho_k = 1 \tag{5.44}$$

3) Marginal weight constraint for combined partition

The marginal weight for a break point in an individual pattern is a linear combination of the marginal weights corresponding to the break points in the combined partition. The validity of these equations has been proved in Appendix A5.

$$\sigma_j^p = \rho_{k(j,p)} + \sum_{k \in K_p(j)} \theta_{kp}\, \rho_k + \sum_{k \in K_p(j+1)} (1-\theta_{kp})\, \rho_k \quad \text{for all } p \in P, \text{ for all } j \in J_p \setminus \{0, n_p\} \tag{5.45a}$$

$$\sigma_j^p = \rho_{k(j,p)} + \sum_{k \in K_p(j+1)} (1-\theta_{kp})\, \rho_k \quad \text{for all } p \in P,\ j = 0 \tag{5.45b}$$

$$\sigma_j^p = \rho_{k(j,p)} + \sum_{k \in K_p(j)} \theta_{kp}\, \rho_k \quad \text{for all } p \in P,\ j = n_p \tag{5.45c}$$

4) Segment-wise adjacency constraint for combined partition

These constraints ensure that we select a single segment in the combined partition and only those marginal weight variables that correspond to the end points of the selected segment can be strictly positive.

$$\sum_{k=1}^{m} R_k = 1 \tag{5.46a}$$

$$\rho_{k-1} + \rho_k \geq R_k \quad \text{for all } k \in \{1,...,m\} \tag{5.47a}$$

$$\rho_0 \leq R_1 \tag{5.47b}$$

$$\rho_k \leq R_k + R_{k+1} \quad \text{for all } k \in \{1,...,m-1\} \tag{5.47c}$$

$$\rho_m \leq R_m \tag{5.47d}$$

5) Binary variable constraint

$$R_k \in \{0,1\} \quad \text{for all } k = 1,...,m \tag{5.48}$$

We call the model formed by the *function value constraint* (Equation 5.1), *the non-negativity constraint* (Equation 5.10), *pattern-based marginal weight constraint* (Equation 5.31), and Equations (5.43–5.48) a *combined partition model with segment-wise adjacency* (CPAR-SEG).

The combined partition model is not only smaller but also tighter than the pattern-based model. Consider the example discussed in Figure 5.5 in which we show an LP solution that is feasible for a pattern-based model but not for a combined partition model, which would have four uniform segments for the X_1 variable (as used in Grid G_2) and will re-distribute the marginal weights of 0.5 along break points 1 and 3 in Grid G_2 over the break points 0, 1, 2 for the X_1 pattern in Grid G_1. This re-distribution, done using Equation (5.36a), will assign marginal weights of 0.25, 0.5 and 0.25 to break points 0, 1 and 2 respectively for the X_1 pattern in Grid G_1, which are different from the current marginal weights of 0.5 each for break points 0 and 4 for the X_1 pattern in Grid G_1. Thus, the LP relaxation value of the combined partition model could be higher than that of a pattern-based model, which implies that a MIP solver might find it easier to solve a combined partition model.

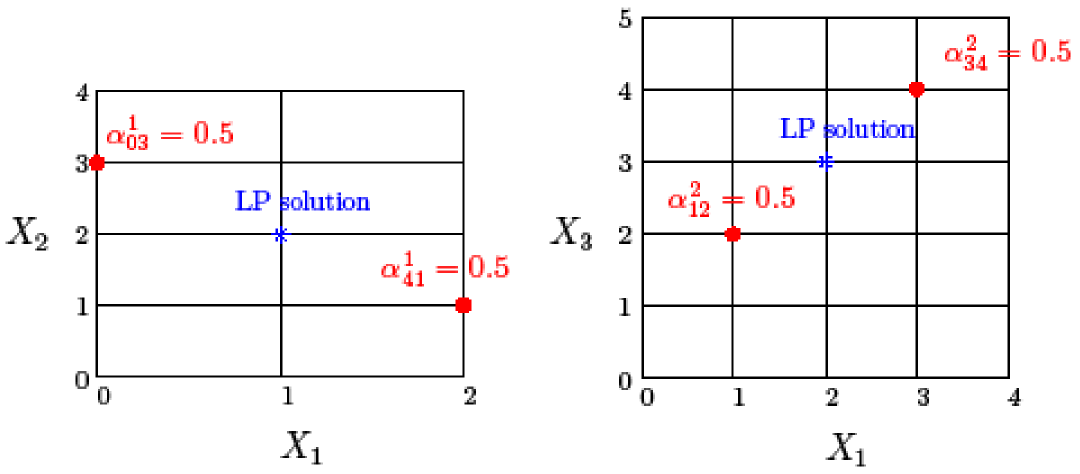


Figure 5.5: Comparing the strength of pattern-based and combined partition models

5.2.3.2 Combined Partition Model with Cumulative Adjacency

As in the other cumulative adjacency modesl, we can use cumulative segment selection and cumulative marginal weight variables for the combined partition and create cumulative adjacency constraints to enforce adjacency in the combined partition.

Let M_k denote the cumulative segment selection (binary) variable for the combined partition that takes the value 1 if the model selects segment k or higher, defined for $k = 1,\ldots,m$ and let μ_k be the cumulative marginal weight for break point k of the combined partition such that $\mu_k = \sum_{k'=k}^{m} \rho_{k'}$ for all $k = 1,\ldots,m$.

We can then have the following cumulative adjacency constraints for the combined partition.

$$\mu_k \le M_k \quad \text{for all } k = 2,\ldots,m \tag{5.49a}$$

$$\mu_k \ge M_{k+1} \quad \text{for all } k = 1,\ldots,m-1 \tag{5.49b}$$

$$M_k \ge M_{k+1} \quad \text{for all } k = 1,\ldots,m-1 \tag{5.49c}$$

$$\mu_0 = 1 \tag{5.50a}$$

$$M_1 = 1 \tag{5.50b}$$

$$M_k \in \{0,1\} \quad \text{for all } k = 1,\ldots,m \tag{5.50c}$$

We also need the *cumulative variable value constraint for combined partition* (Equation 5.51), and the *cumulative marginal weight constraint for combined partition* (Equation 5.52).

$$X = \sum_{k=0}^{m-1} x_k(\mu_k - \mu_{k+1}) + x_m\mu_m \tag{5.51}$$

$$\begin{aligned}\sigma_j^p = \left(\mu_{k(j,p)} - \mu_{k(j,p)+1}\right) + \sum_{k\in K_p(j)} \theta_{kp}\left(\mu_{k(j,p)} - \mu_{k(j,p)+1}\right)\\ + \sum_{k\in K_p(j+1)} (1-\theta_{kp})\left(\mu_{k(j,p)} - \mu_{k(j,p)+1}\right) \quad \forall\, p \in P,\ j \in J_p \setminus \{0, n_p\}\end{aligned} \tag{5.52a}$$

$$\sigma_j^p = \left(\mu_{k(j,p)} - \mu_{k(j,p)+1}\right) + \sum_{k\in K_p(j+1)} (1-\theta_{kp})\left(\mu_{k(j,p)} - \mu_{k(j,p)+1}\right) \forall\, p \in P,\ j = 0 \tag{5.52b}$$

$$\sigma_j^p = \left(\mu_{k(j,p)} - \mu_{k(j,p)+1}\right) + \sum_{k\in K_p(j)} \theta_{kp}\left(\mu_{k(j,p)} - \mu_{k(j,p)+1}\right) \forall\, p \in P,\ j = n_p \tag{5.52c}$$

We call the model formed by the *function value constraint* (Equations 5.1), *the non-negativity constraint* (Equation 5.10), the *pattern-based marginal weight constraint* (Equation 5.31) and Equations (5.49–5.52) a *combined partition model with cumulative adjacency* (CPAR-CUM).

5.2.3.3 Combined Partition Model with Logarithmic Indexing

We could also enforce adjacency in the combined partition by using a logarithmic indexing scheme. To each of the *m* segments in the combined partition, let us assign a reflected binary code comprised of $\text{ceil}(\log_2(m))$ bits. Let $\overline{K}(q)$ be the set of break points in the combined partition that serve as the vertex of at least one cell whose reflected binary code contains a zero at the position *q*, and $\hat{K}(q)$ be the set of break points in the combined partition that serve as the vertex of at least one cell whose reflected binary code contains a one at position *q*. Let us define binary variables $D_q, q = 1,\ldots,\log_2(m)$ such that D_q takes the value one if we select a cell whose reflected binary code contains a one at position *q* and D_q takes the value zero if we select a cell whose reflected binary code contains a zero at position *q*. The logarithmic adjacency conditions for the combined partition are as follows.

$$\sum_{k \notin \overline{K}(q)} \rho_k \le D_q \quad \text{for } q = 1, 2, \ldots, \text{ceil}[\log_2(m)] \tag{5.53a}$$

$$\sum_{k \notin \hat{K}(q)} \rho_k \le 1 - D_q \quad \text{for } q = 1, 2, \ldots, \text{ceil}[\log_2(m)] \tag{5.53b}$$

$$D_q \in \{0,1\} \text{ for } q =, 2, \ldots, \text{ceil}[\log_2(m)] \tag{5.54}$$

Equation (5.53a) ensures that if we select a segment in the combined partition whose reflected binary code has a zero at position *k* (i.e., if D_q is zero), then all the break points that do not belong to $\overline{K}(q)$ are assigned a weight of zero. On the other hand, Equation (5.53b) ensures that if we select a segment in the combined partition whose reflected binary value has a one at position *k* (i.e., if D_q is one), then all the break points that do not belong to $\hat{K}(q)$ have a weight of zero.

We call the model formed by the *function value constraint* (Equation 5.1), *the non-negativity constraint* (Equation 5.10), *pattern-based marginal weight constraint* (Equation 5.31), the variable value constraint for combined partition (Equation 5.43), the convex combination constraint for combined partition (Equation 5.44), the marginal weight constraint for combined partition (Equation 5.45), and Equations (5.53–5.54) a *combined partition model with logarithmic adjacency* (CPAR-LOG).

5.2.3.4 Combined Partition Model with SOS-2-based Adjacency

We could also enforce adjacency in the combined partition by defining the marginal weight variables in the combined partition as SOS-2 (Equation 5.55).

$$\rho_0,\ldots,\rho_m \ \text{SOS}-2 \qquad (5.55)$$

We call the model formed by the *function value constraint* (Equation 5.1), *the non-negativity constraint* (Equation 5.10), *pattern-based marginal weight constraint* (Equation 5.31), the variable value constraint for combined partition (Equation 5.43), the convex combination constraint for combined partition (Equation 5.44), the marginal weight constraint for combined partition (Equation 5.45), and Equations (5.55) a *combined partition model with SOS-2 adjacency* (CPAR-SOS).

The notation used in the four types of combined partition models is summarized in Appenix A12.

5.2 Problem Strengthening using Constraint-based Inequalities

If a nonlinear program contains constraints involving two or more variables that also appear as independent variables of the same or different nonlinear functions, then we can strengthen the PLA-based mixed integer program by adding valid inequalities which relate the segment-selection and marginal weight variables corresponding to those independent variables. These valid inequalities arise beacuse the valid segment-selection and marginal weight variables corresponding to the independent variables are now dependent on one another. These valid inequalities strengthen the LP relaxation value of the PLA-based mixed integer program and as shown in the computational results, help us tackle some of the more difficult problems in our problem set.

Consider a situation in which we have a nonlinear program with a linear constraint involving variables *X* and *Y* such that *X* and *Y* also appear as independent variables either in the same nonlinear function or in two different nonlinear functions. We can have four types of linear constraints involing *X* and *Y*.

1. $Y \leq aX + b$: In this case, an upper bound on *X* induces an upper bound on *Y*.
2. $Y \geq -aX + b$: In this case, an upper bound on *X* induces a lower bound on *Y*.
3. $Y \leq -aX + b$: In this case, a lower bound on *X* induces an upper bound on *Y*.
4. $Y \geq aX + b$: In this case, a lower bound on *X* induces a lower bound on *Y*.

We now discuss each of these four cases sepately and show how these induced bounds can be used to generate valid inequalities relating the binary and marginal weight variables for the *X* and *Y* patterns.

Notation

I:	set of break points in the X partition
J:	set of break points in the Y partition
i:	index for break points and segments for the X partition
j:	index for break points and segments for the Y partition
n_X :	number of segments in the X partition
n_Y :	number of segments in the Y partition
σ_i^X	marginal weight variable for break point i along the X partition
τ_i^X	cumulative marginal weight variable for break point i along the X partition
σ_j^Y	marginal weight variable for break point j along the Y partition
τ_j^Y	cumulative marginal weight variable for break point j along the X partition
S_i^X	segment selection variable for segment i along the X partition, which is one if we select segment i and is zero otherwise
T_i^X	segment selection variable for segment i along the X partition, which is one if we select a segment with index i or higher, and is zero otherwise
S_j^Y	segment selection variable for segment j along the Y partition, which is one if we select segment j and is zero otherwise
T_j^Y	segment selection variable for segment j along the Y partition, which is one if we select a segment with index j or higher, and is zero otherwise

5.2.1 Upper Bound on a Variable Inducing an Upper Bound on another Variable

Consider the situation where *X* and *Y* are present in two different nonlinear functions but are related by the constraint $Y \le aX + b$. In this situation, an upper bound on *X* induces an upper bound on *Y*. Let us assume that we select an *X* segment with index *i* or lower. This implies that *X* cannot exceed x_i. Let $j(i) = \min j : ax_i + b \ge y_j$. Then, we cannot select a *Y* segment whose index is greater than $j(i)$. This gives us the following valid inequalities.

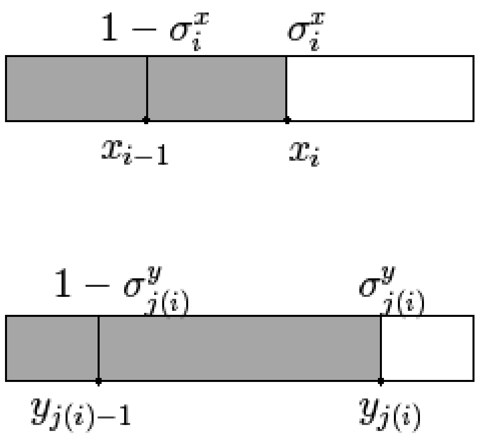


Figure 5.6: Upper bound on X inducing an upper bound on Y

a) Inequalities relating binary variables

Let $\bar{I}_1$ be the set of break points in the *X* pattern such that $I_1 = \{i : 1 \le j(i) \le n_Y - 1\}$. Then, if we select an *X* segment with index *i* or lower, then we must select a *Y* segment with index $j(i)$ or less. So, we have the following inequality.

$$1 - T_{j(i)+1}^{Y} \ge 1 - T_{i+1}^{X} \text{ for every } i \in I_1 \tag{5.56}$$

b) Inequalities relating marginal weight variables

Let $\bar{y}_i = ax_i + b$. Then, if *X* has an upper bound of x_i, *Y* has an upper bound of $\bar{y}_i$. This implies that there is an upper bound on the marginal weight variable for break point *j*(*i*) along the *Y* partition.

Consider a case where we want to choose

- segment *i* for the *X* variable with weights σ_i^X and $1-\sigma_i^X$ assigned to break points *i* and *i*-1, and
- segment *j*(*i*) for the *Y* variable with weights $\sigma_{j(i)}^Y$ and $1-\sigma_{j(i)}^Y$ assigned to break points *j*(*i*) and *j*(*i*)-1.

Since $Y \le aX + b$, we get the following.

$$\sigma_{j(i)}^Y y_{j(i)} + (1-\sigma_{j(i)}^Y) y_{j(i)-1} \le a(\sigma_i^X x_i + (1-\sigma_i^X) x_{i-1}) + b \tag{5.57}$$

$$\sigma_{j(i)}^Y \le \sigma_i^X (p_i - q_i) + q_i \text{ where } p_i = \frac{\bar{y}_i - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}} \text{ and } q_i = \frac{\bar{y}_{i-1} - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}}$$

$$\tau_{j(i)}^Y \le \tau_i^X (p_i - q_i) + q_i \text{ since in this case } \sigma_i^X = \tau_i^X \text{ and } \sigma_{j(i)}^Y = \tau_{j(i)}^Y \tag{5.58}$$

Thus, if we select segment *i* for the *X* variable and assign a cumulative weight of τ_i^X to break point *i*, then we have an upper bound on the cumulative weight variable for break point *j*(*i*). This leads us to the following inequality.

$$\tau_{j(i)}^Y \le \tau_i^X (p_i - q_i) + q_i (1-T_{i+1}^X) - q_i (1-T_{j(i)}^Y) + T_{i+1}^X \tag{5.59}$$

The proof of the validity of this inequality is provided in Appendix A6.

5.2.2 Upper Bound on a Variable Inducing a Lower Bound on another Variable

Consider the situation where *X* and *Y* are present in two different nonlinear functions but are related by the constraint $Y \geq -aX + b$. In this situation, an upper bound on *X* induces a lower bound on *Y*. Let us assume that we select an *X* segment with index *i* or lower. This implies that *X* cannot exceed x_i. Let $j(i) = \max j : -ax_i + b \leq y_j$. Then, we cannot select a *Y* segment whose index is less than $j(i)+1$. This gives us the following valid inequalities.

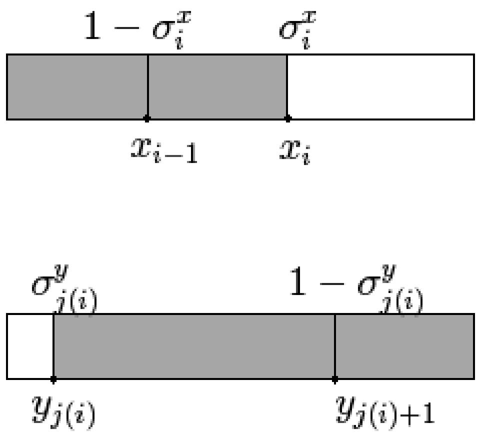


Figure 5.7: Upper bound on X inducing a lower bound on Y

a) Inequalities relating binary variables

Let $\bar{I}_2$ be the set of break points in the *X* pattern such that $I_2 = \{i : 2 \leq j(i) \leq n_Y - 1\}$. Then, if we select an *X* segment with index *i* or lower, then we must select a *Y* segment with index $j(i)+1$ or higher. So, we have the following inequality.

$$T_{j(i)+1}^{Y} \geq 1 - T_{i+1}^{X} \quad \text{for every } i \in I_2 \qquad (5.60)$$

b) Inequalities relating marginal weight variables

Let $\bar{y}_i = -ax_i + b$. Then, if *X* has an upper bound of x_i, *Y* has a lower bound of $\bar{y}_i$. This implies that there is an upper bound on the marginal weight variable for break point *j*(*i*) along the *Y* partition, and consequently a lower bound on the cumulative marginal weight for break point $j(i)+1$ along the *Y* partition. Consider a case where we want to choose

- segment *i* for the *X* variable with weights σ_i^X and $1-\sigma_i^X$ assigned to break points *i* and *i*-1
- segment $j(i)+1$ for the *Y* variable with weights $\sigma_{j(i)}^Y$ and $1-\sigma_{j(i)}^Y$ assigned to break points $j(i)$ and $j(i)+1$

Since $Y \geq -aX + b$, we get the following.

$$\sigma_{j(i)}^Y y_{j(i)} + (1-\sigma_{j(i)}^Y) y_{j(i)+1} \geq -a(\sigma_i^X x_i + (1-\sigma_i^X) x_{i-1}) + b \tag{5.61}$$

$$\sigma_{j(i)}^Y \leq \sigma_i^X (p_i - q_i) + q_i \text{ where } p_i = \frac{y_{j(i)+1} - \bar{y}_i}{y_{j(i)+1} - y_{j(i)}} \text{ and } q_i = \frac{y_{j(i)+1} - \bar{y}_{i-1}}{y_{j(i)+1} - y_{j(i)}}$$

$$1-\tau_{j(i)+1}^Y \leq \tau_i^X (p_i - q_i) + q_i \text{ since } \sigma_i^X = \tau_i^X \text{ and } \sigma_{j(i)}^Y = 1-\tau_{j(i)+1}^Y$$

$$\tau_{j(i)+1}^Y \geq (1-q_i) - \tau_i^X (p_i - q_i) \tag{5.62}$$

Thus, if we select segment *i* for the *X* variable and assign a cumulative weight of τ_i^X to break point *i*, then we have a lower bound on the cumulative weight variable for break point $j(i)+1$. This leads us to the following inequality.

$$\tau_{j(i)+1}^Y \geq (1-q_i)(1-T_{i+1}^X) - \tau_i^X (p_i - q_i) + q_i T_{j(i)+2}^Y \tag{5.63}$$

The proof of the validity of this inequality is provided in Appendix A6.

5.2.3 Lower Bound on a Variable Inducing an Upper Bound on another Variable

Consider the situation where *X* and *Y* are present in two different nonlinear functions but are related by the constraint $Y \le -aX + b$. In this situation, a lower bound on *X* induces an upper bound on *Y*. Let us assume that we select an *X* segment with index $i+1$ or lower. This implies that *X* has a lower bound of x_i. Let $j(i) = \min\ j : -ax_i + b \ge y_j$. Then, we cannot select a *Y* segment whose index is greater than $j(i)$. This gives us the following valid inequalities.

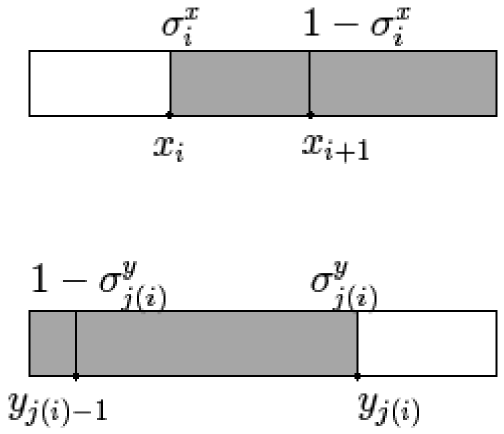


Figure 5.8: Lower bound on X inducing an upper bound on Y

a) Inequalities relating binary variables

Let $\bar{I}_3$ be the set of break points in the *X* partition such that $I_3 = \{i : 1 \le j(i) \le n_Y - 1\}$. Then, if we select an *X* segment with index $i+1$ or higher, then we must select a *Y* segment with index $j(i)$ or lower. So, we have the following inequality.

$$1 - T_{j(i)+1}^{Y} \ge T_{i+1}^{X} \text{ for every } i \in I_3 \tag{5.64}$$

b) Inequalities relating marginal weight variables

Let $\bar{y}_i = -ax_i + b$. Then, if *X* has a lower bound of x_i, *Y* has an upper bound of $\bar{y}_i$. This implies that there is an upper bound on the marginal weight variable for break point *j*(*i*) along the *Y* partition, and consequently an upper bound on the cumulative marginal weight for break point $j(i)$ along the *Y* partition. Consider a case where we want to choose

- segment *i*+1 for the *X* variable with weights σ_i^X and $1-\sigma_i^X$ assigned to break points *i* and *i*+1, and
- segment $j(i)$ for the *Y* variable with weights $\sigma_{j(i)}^Y$ and $1-\sigma_{j(i)}^Y$ assigned to break points $j(i)$ and $j(i)-1$.

Since $Y \le -aX + b$, we get the following.

$$\sigma_{j(i)}^Y y_{j(i)} + (1-\sigma_{j(i)}^Y) y_{j(i)-1} \le -a(\sigma_i^X x_i + (1-\sigma_i^X) x_{i+1}) + b \tag{5.65}$$

$$\sigma_{j(i)}^Y \le \sigma_i^X \left(p_i - q_i\right) + q_i \text{ where } p_i = \frac{\bar{y}_i - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}} \text{ and } q_i = \frac{\bar{y}_{i+1} - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}}$$

$$\tau_{j(i)}^Y \le (1-\tau_{i+1}^X)\left(p_i - q_i\right) + q_i \text{ since } \sigma_i^X = 1-\tau_{i+1}^X \text{ and } \sigma_{j(i)}^Y = \tau_{j(i)}^Y$$

$$\tau_{j(i)}^Y \le p_i - \tau_{i+1}^X\left(p_i - q_i\right) \tag{5.66}$$

Thus, if we select segment $i+1$ for the *X* variable and assign a cumulative weight of τ_{i+1}^X to break point $i+1$, then we have an upper bound on the cumulative weight variable for break point $j(i)$. This leads us to the following inequality.

$$\tau_{j(i)}^Y \le p_i T_{i+1}^X - (p_i - q_i)\tau_{i+1}^X - q_i(1-T_{j(i)}^Y) + (1-T_{i+1}^X) \tag{5.67}$$

The proof of the validity of this inequality is provided in Appendix A6.

5.2.4 Lower Bound on a Variable Inducing a Lower Bound on another Variable

Consider the situation where *X* and *Y* are present in two different nonlinear functions but are related by the constraint $Y \geq aX + b$. In this situation, a lower bound on *X* induces a lower bound on *Y*. Let us assume that we select an *X* segment with index $i+1$ or higher. This implies that *X* cannot be lower than x_i. Let $j(i) = \max j : ax_i + b \leq y_j$. Then, we cannot select a *Y* segment whose index is less than $j(i)+1$. This gives us the following valid inequalities.

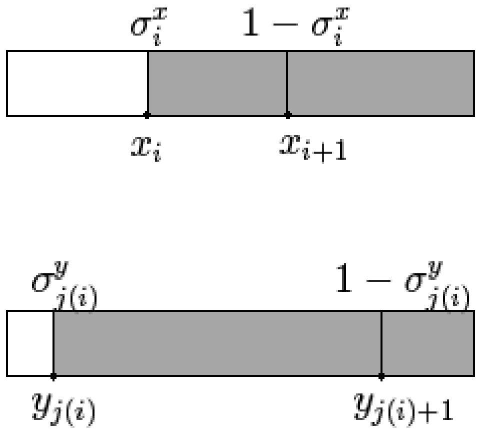


Figure 5.9: Lower bound on X inducing a lower bound on Y

a) Inequalities relating binary variables

Let $\bar{I}_4$ be the set of break points in the *X* pattern such that $I_4 = \{i : 1 \leq j(i) \leq n_Y - 1\}$. Then, if we select an *X* segment with index $i+1$ or higher, then we must select a *Y* segment with index $j(i)+1$ or higher. So, we have the following inequality.

$$T_{j(i)+1}^{Y} \geq T_{i+1}^{X} \text{ for every } i \in I_1 \tag{5.68}$$

b) Inequalities relating marginal weight variables

Let $\bar{y}_i = ax_i + b$. Then, if X has a lower bound of x_i, Y has a lower bound of $\bar{y}_i$. This implies that there is an upper bound on the marginal weight variable for break point $j(i)$ along the Y partition and consequently a lower bound on the cumulative marginal weight for break point $j(i)+1$ along the Y partition. Consider a case where we want to choose

- segment $i+1$ for the X variable with weights σ_i^X and $1-\sigma_i^X$ assigned to break points i and i+1
- segment $j(i)+1$ for the Y variable with weights $\sigma_{j(i)}^Y$ and $1-\sigma_{j(i)}^Y$ assigned to break points $j(i)$ and $j(i)+1$.

Since $Y \geq aX + b$, we get the following.

$$\sigma_{j(i)}^Y y_{j(i)} + (1-\sigma_{j(i)}^Y) y_{j(i)+1} \geq a(\sigma_i^X x_i + (1-\sigma_i^X) x_{i+1}) + b \tag{5.69}$$

$$\sigma_{j(i)}^Y \leq \sigma_i^X (p_i - q_i) + q_i \text{ where } p_i = \frac{y_{j(i)+1} - \bar{y}_i}{y_{j(i)+1} - y_{j(i)}} \text{ and } q_i = \frac{y_{j(i)+1} - \bar{y}_{i+1}}{y_{j(i)+1} - y_{j(i)}}$$

$$1-\tau_{j(i)+1}^Y \leq (1-\tau_{i+1}^X)(p_i - q_i) + q_i \text{ since in this case } \sigma_i^X = 1-\tau_{i+1}^X \text{ and } \sigma_{j(i)}^Y = 1-\tau_{j(i)+1}^Y$$

$$1-\tau_{j(i)+1}^Y \leq p_i - q_i - \tau_{i+1}^X (p_i - q_i) + q_i$$

$$\tau_{j(i)+1}^Y \geq (1-p_i) + (p_i - q_i)\tau_{i+1}^X \tag{5.70}$$

Thus, if we select segment $i+1$ for the X variable and assign a cumulative weight of τ_{i+1}^X to break point $i+1$, then we have an upper bound on the cumulative weight variable for break point $j(i)+1$ along the Y partition. This leads us to the following inequality.

$$\tau_{j(i)+1}^Y \geq (p_i - q_i)\tau_{i+1}^X + (1-p_i) T_{i+1}^X + q_i T_{j(i)+2}^Y \tag{5.71}$$

The proof of the validity of this inequality is provided in Appendix A6.

5.3 PROBLEM STRENGTHENING USING BOUNDS

5.3.1 Problem Strengthening Using Variable Bounds

If a nonlinear program contains a nonlinear function such that the function value has an externally specified upper or lower bound, then there might be domain points that violate these bounds and therefore can never be present in a feasible solution to the PLA-based mixed integer program. In such a case, we can remove from the PLA-based mixed integer program the vertex weight variables corresponding to these infeasible domain points. Before discussing how to do so, we need to define some notation.

Notation

V	set of vertices in the grid used for the piecewise linear approximation of the nonlinear functions
C	set of cells formed by the grid
v, v'	index over V
c	index over C
$V(c)$	set of vertices for cell c
$C(v)$	set of cells with a vertex at point v
$f(.)$	nonlinear function that is being approximated
L, U	exogenous lower and upper bounds on $f(.)$ either explicitly specified in the nonlinear program or implied by some constraint
f_v	function value at vertex point v
$N(v)$	set of vertices (excluding v) with which v can form a convex combination, given by $\bigcup_{c \in C(v)} V(c) \setminus v$

$f_v^{\min}$ Minimum function value across all vertices with which vertex v can form a convex combination, i.e. , $f_v^{\min} = \mathrm{Min}\, f_{v'} : v' \in N(v)$

$f_v^{\max}$ Maximum function value across all vertices with which vertex v can form a convex combination, i.e. , $f_v^{\max} = \mathrm{Max}\, f_{v'} : v' \in N(v)$

α_v vertex weight variable for vertex v

We now consider two scenarios in which a vertex might have an associated function value that violates the externally specified bounds for that variable.

1) Vertex with function value less than L

Consider a vertex v such that f_v is less than L. It is possible that this vertex could form a convex combination with another vertex v' belonging to $N(v)$. We can have two cases.

<u>*Case 1*</u>*:* All vertices belonging to $N(v)$ have a function value less than L. For example, the vertex P_2 in Figure 5.10 has a function value less than L and all the neighboring vertices with which it can form a convex combination have also a function value less than L. Therefore, the convex combination of vertex v with any neighboring vertex cannot be feasible and so we can remove the vertex weight variable associated with vertex v in the PLA-based mixed integer program.

<u>*Case 2*</u>: One or more vertices belonging to $N(v)$ have a function value greater than L. For example, the vertices P_1 and P_3 in Figure 5.10 have a function value less than L but they can form a convex combination with one of their neighbors such the value of this convex combination is less than L. Therefore, such vertices can have a positive vertex weight in the MIP solution.

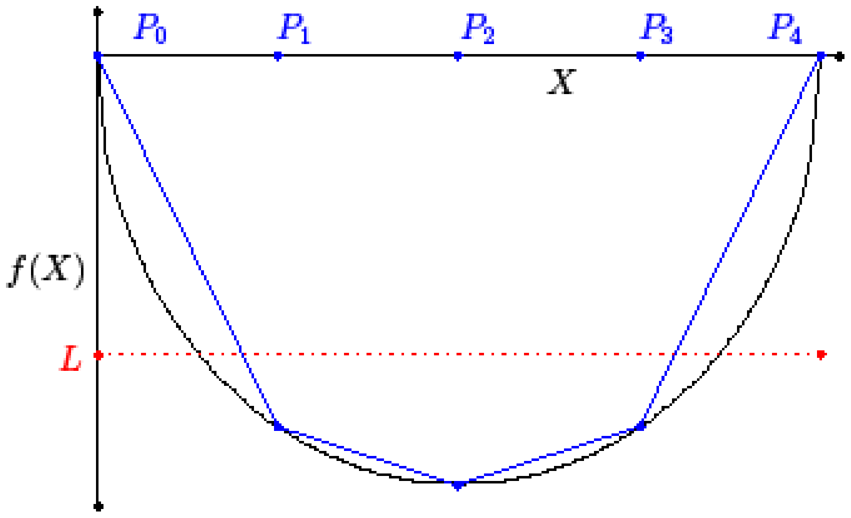


Figure 5.10: Bounding vertex weight variables using an external lower bound

However, if a vertex *v* has a function value less than *L* but it can form a combination with a neighboring vertex with a function value less than *L* then we can obtain an upper bound on the vertex weight variable associated with *v* as given by Equation (5.72).

$$\alpha_v \leq \frac{f_v^{\max} - L}{f_v^{\max} - f_v} \tag{5.72}$$

<u>Proof</u>: Let us assume that the solution to the PLA-based mixed integer program chooses vertex *v* (with weight α_v) and distributes a weight of (1-α_v) across other vertices belonging to *N*(*v*). Then, if there is at least one vertex in the set *N*(*v*) with a function value greater than *L*, then we have the following.

$$\alpha_v f_v + \sum_{v' \in N(v)} \alpha_{v'} f_{v'} \geq L$$

$$\alpha_v f_v + \sum_{v' \in N(v)} \alpha_{v'} f_v^{\max} \geq L$$

$$\alpha_v f_v + f_v^{\max}(1 - \alpha_{v'}) \geq L$$

$$\alpha_v \leq \frac{f_v^{\max} - L}{f_v^{\max} - f_v}$$

2) Vertex with function value greater than *U*

Consider a vertex *v* such that f_v is greater than *U*. This vertex could form a convex combination with another vertex *v'* belonging to *N* (*v*). We can have two cases.

*Case 1*: All vertices belonging to *N* (*v*) have a function value greater than *U*. Therefore, *v* cannot form a feasible convex combination with any vertex in *N* (*v*) and we can remove the vertex weight variable associated with vertex *v* in the PLA-based mixed integer program. For example, vertex P_2 in Figure 5.11 cannot have a positive vertex weight because it cannot form a feasible convex combination with any of its neighbors.

*Case 2*: One or more vertices belonging to *N* (*v*) have a function value less than *U*. Therefore, *v* can form a feasible convex combination with some vertex and can have a positive vertex weight in the MIP solution. For example, vertices P_1 and P_3 can form a feasible convex combination with one of their neighbors. In this case, we can have an upper bound on the vertex weight variable associated with *v* given by Equation (5.73).

$$\alpha_v \leq \frac{U - f_v^{\min}}{f_v - f_v^{\min}} \tag{5.73}$$

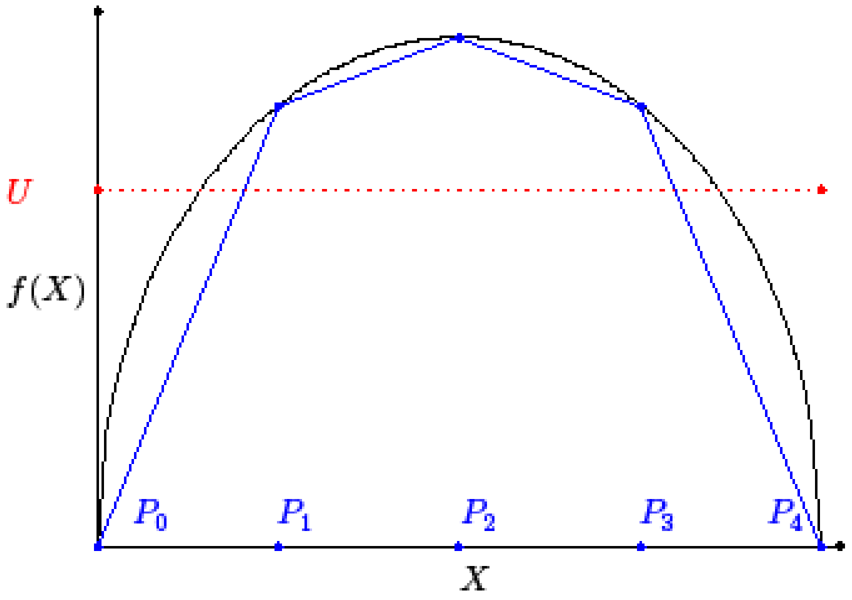


Figure 5.11: Bounding vertex weight variables using an external upper bound

Proof: Let us assume that the solution to the PLA-based mixed integer program chooses vertex v (with weight α_v) and distributes a weight of (1-α_v) across other vertices belonging to $N(v)$. Then, if there is at least one vertex in the set $N(v)$ with a function value less than U, then we have the following.

$$\alpha_v f_v + \sum_{v' \in N(v)} \alpha_{v'} f_{v'} \leq U$$

$$\alpha_v f_v + \sum_{v' \in N(v)} \alpha_{v'} f_v^{\min} \leq U$$

$$\alpha_v f_v + f_v^{\min}(1 - \alpha_v) \leq U$$

$$\alpha_v \leq \frac{U - f_v^{\min}}{f_v - f_v^{\min}}$$

5.3.2 Problem Strengthening Using Constraint-based Bounds

Consider a situation where we have a non-linear program with (a) an actual or an implied nonlinear function $f(X_1, X_2, \mathbf{Y})$ involving variables X_1, X_2 and $\mathbf{Y}$, where $\mathbf{Y}$ is a (possible empty) vector of variables, and (b) a constraint $h(X_1, X_2) \leq 0$, which can be linear or nonlinear. We call $h(X_1, X_2) \leq 0$ an implied nonlinear function if it is not present in the original nonlinear program but has been derived from other constraints in the nonlinear program. Consider the grid used to create a piecewise linear approximation of f. If there are vertices in this grid that do not satisfy the constraint $h(X_1, X_2) \leq 0$, then using a reasoning similar to that in the previous section, we can come up with a bound on the vertex weight variables.

We first define some additional notation. Let h_v be the value of *h* at vertex point *v* and $h_v^{\min}$ be the smallest value of *h* across all vertices with which vertex *v* can form a convex combination, i.e. , $h_v^{\min} = \mathrm{Min}\, h_{v'} : v' \in N(v)$.

Consider a vertex *v* such that h_v is greater than 0. This vertex could form a convex combination with another vertex *v'* belonging to *N* (*v*). We can have two cases.

<u>*Case 1*</u>: All vertices *v'* belonging to *N* (*v*) have an $h_{v'}$ value greater than 0. Therefore, *v* cannot form a feasible convex combination with any vertex in *N* (*v*) and we can remove the vertex weight variable associated with vertex *v* in the PLA-based mixed integer program.

<u>*Case 2*</u>: One or more vertices *v'* belonging to *N* (*v*) have an $h_{v'}$ value less than 0. Therefore, *v* can form a feasible convex combination with some vertex and can have a positive vertex weight in the MIP solution. In this case, we can have an upper bound on

the vertex weight variable associated with *v* given by Equation (5.74). We could prove this result by the same reasoning that we used in the proofs in Section 5.3.1.

$$\begin{aligned} \alpha_v &\le \frac{0 - h_v^{\min}}{h_v - h_v^{\min}} \\ \alpha_v &\le -\frac{h_v^{\min}}{h_v - h_v^{\min}} \end{aligned} \tag{5.74}$$

Chapter 6: Computational Results

6.1 Goals

The aim of the computations is to demonstrate the effectiveness of the PLA approach and of the various enhancements in terms of problem reformulation, grid design, and MIP modeling.

1) Problem reformulation: We show how different reformulations of the nonlinear program result in different MIP solutions and thereby different local solver solutions.
2) Grid design: We show how the absolute size, the relative size and the shape of the cells affect the performance of the PLA approach.
3) MIP modeling: We illustrate the effectiveness of three types of MIP modeling enhancements: model formulation, model reduction and model strengthening.
    a. Model formulation-based enhancements involve using different MIP representations of a piecewise linear function. Depending upon how adjacency conditions are imposed in a model, we can have four types of model formulations.

        (i) Models with s*egmentwise non-logarithmic adjacency* involve defining adjacency conditions for each segment along each dimension of a non-linear function.

        (ii) Models with c*umulative non-logarithmic adjacency* have adjacency constraints defined for groups of contiguous segments along each dimension of a nonlinear function.

        (iii) Models with l*ogarithmic adjacency* use a logarithmic indexing scheme and corresponding adjacency conditions.

(iv) Models with *SOS-2-based adjacency* enforce adjacency by defining certain variables as SOS-2.

b. Model reduction techniques seek to decrease the size of the PLA-based mixed integer program by using variable bounds

c. Model strengthing techniques seek to increase the LP relaxation value of the mixed integer program by (i) creating stronger models based on the presence of common variables in different nonlinear functions, and (ii) adding valid inequalities based on the relationships between different variables that exist within one or more nonlinear functions. For the former, we have three different model types: (i) *individual grid* model in which we have a PLA-based MIP model for each variable in each non-linear function in the nonlinear program, (ii) *pattern-based* model, in which we have an individual PLA-based MIP model for each pattern and variable combination, and (iii) *combined partition* model, in which we have a single PLA-based MIP model for each variable that appears in some nonlinear function in the non-linear program. As mentioned earlier, a pattern is a partition induced by a set of break points along the domain of a function. For the latter, we have model strengthening based on valid inequalities that arise when the nonlinear program has constraints that link two or more variables present in one or more (linear or nonlinear) functions in the nonlinear program.

To demonstrate the effectiveness of the PLA approach and of the enhancements based on problem reformulation, grid design, and MIP modeling, we use the following sets of problems.

1) 55 Unconstrained problems from Ali et al. (2005)
2) 13 pooling problems from Adhya et al. (1999)
3) 48 constrained nonlinear problems with continuos variables from Global-Lib, an online repository available at *http://www.gamsworld.org/global/globallib.htm*.
4) 22 constrained mixed integer nonlinear problems from MINLP-Lib, an online repository available at *http://www.gamsworld.org/minlp/minlplib/minlpstat.htm*.

Table 6.1 provides a summary of how these problem sets will be used to accomplish these goals.

Section	Goal	Problem Sets
6.2	Introduction	-
6.3	Apply the basic PLA method	Ali et al., Global-Lib, Pooling, MINLP-Lib
6.4	Effect of increased grid resolution	Ali et al. Global-Lib, MINLP-Lib
6.5	Effect of non-uniform grids	Ali et al., Global-Lib
6.6	Effect of grid shape	Pooling
6.7	Effect of indexing scheme	Pooling
6.8	Comparison with CONOPT	Ali et al., Global-Lib, Pooling
6.9	Comparing grid-based and pattern-based models	MINLP-Lib
6.10	Comparing pattern-based and combined partition models	MINLP-Lib
6.11	Effect of problem reduction and strengthening strategies	MINLP-Lib
6.12	Comparison with DICOPT	MINLP-Lib

Table 6.1: Computational design

6.2 Introduction

We now discuss the process that we used to apply the PLA method to the various problem sets. This discussion includes (a) the hardware and software specifications of the environment in which these computations have been performed, (b) the different metrics that we have used to measure the quality of the PLA-based solutions, and (c) the guidelines that we have used to reformulate the nonlinear programs into a form that the PLA method can work on.

6.2.1 Hardware and Software Specifications

For our runs, we used a Windows machine with a RAM of 32 GB and a 3.40 GHz Intel Core *i*7-3770 processor. We implemented the PLA algorithm as a C callable library that (a) reads from a grid specification file the set of variables, linear constraints, nonlinear functions and grid specifications for each of the nonlinear functions in the nonlinear program, (b) generates a mixed integer model, (c) solves the mixed integer program to optimality using Cplex 12.6 *C*-callable library, and (d) passes the MIP solution to CONOPT (version 3.15N), which acts as the local nonlinear solver.

The grid specification file contains the following information for each nonlinear function that exists in the nonlinear program.

1) Grid dimension
2) Grid shape
3) No of segments along each dimension
4) Whether the grid is uniform or non-uniform
5) No. of intervals along each dimension (for non-uniform grids)
6) Error metrics for the shortest path (for non-uniform grids)

7) Number of sample points for estimating the arc costs (for non-uniform grids)

To apply the PLA method to mixed integer nonlinear programs, we fix the integer variables based on the MIP solution thereby creating a continuous nonlinear program that can be solved using CONOPT.

We used the following specifications for Cplex.

1) Presolve: On
2) MIP emphasis : Balance optimality and feasibility
3) MIP search method: Dynamic search
4) MIP termination criteria: CPU time of 600 seconds or 0% integrality gap

6.2.2 Metrics for Measuring MIP Solution Quality

We use the following metrics for assessing the quality of the MIP solution.

1) *Relative MIP-global distance* (δ) quantifies how close the MIP solution is to the globally optimal solution. If $\mathbf{x_g}$ and $\mathbf{x_m}$ denote the global and MIP solutions, then δ is defined as follows.

$$\delta = \begin{cases} \dfrac{\| \mathbf{x_m} - \mathbf{x_g} \|}{\| \mathbf{x_g} \|} \times 100\% & \text{if } \mathbf{x_g} \neq \mathbf{0} \\ 0\% & \text{if } \mathbf{x_g} = \mathbf{0},\ \mathbf{x_m} = \mathbf{x_g} \\ 100\% & \text{if } \mathbf{x_g} = \mathbf{0},\ \mathbf{x_m} \neq \mathbf{x_g} \end{cases}$$

For our computations, a value less than 10^{-6} is treated as zero.

2) *Relative MIP approximation error* (σ) quantifies the difference between the MIP objective value as given by the MIP solver and the true function value of the MIP solution. The true function value of the MIP solution is the solution's objective function value in the original non-linear program. If z_m^{sol} denotes the objective value

of the MIP as given by the solver and z_m denotes the true value of the MIP solution, then σ is defined as follows.

$$\sigma = \begin{cases} \dfrac{|z_m - z_m^{sol}|}{|z_m|} \times 100\% & \text{if } z_m \neq 0 \\ 0\% \quad \text{if } z_m = 0, z_m^{sol} = z_m \\ 100\% \quad \text{if } z_m = 0, z_m^{sol} \neq z_m \end{cases}$$

3) *Relative MIP-global gap* (μ) indicates how close the true value of the MIP solution is to the objective value of the global solution. If z_g denotes the objective value of the global solution and z_m denotes the true cost of the MIP solution, then μ is defined as follows.

$$\mu = \begin{cases} \dfrac{|z_m - z_g|}{|z_g|} \times 100\% & \text{if } z_g \neq 0 \\ 0\% \quad \text{if } z_g = 0, z_m = z_g \\ 100\% \quad \text{if } z_g = 0, z_m \neq z_g \end{cases}$$

4) *Fraction of feasible constraints* (φ) indicates the fraction of constraints in the original nonlinear program that are satisfied by the MIP solution. This metric is used only for constrained optimization problems.

   ϕ *= No. of constraints satisfied by MIP solution / Total no. of original constraints*

5) *Fraction of correctly identified integer variables* (ψ) indicates the fraction of integer variables in the original nonlinear program that have the same value in the MIP solution as in the globally optimal solution to the original nonlinear program. This metric is used only for mixed integer nonlinear problems.

   ψ *= No. of correctly identified integer variables / Total no. of integer variables*

6) *Relative local-global gap* (η), which indicates how close the objective value of the final local solution (found by the local nonlinear solver) is to the objective value of the global solution. If z_g denotes the objective value of the global solution and z_n denotes the objective value of the final local (nonlinear) solution, then η is defined as follows.

$$\eta = \begin{cases} \dfrac{|z_n - z_g|}{|z_g|} \times 100\% & \text{if } z_g \neq 0 \\ 0\% & \text{if } z_g = 0, z_n = z_g \\ 100\% & \text{if } z_g = 0, z_n \neq z_g \end{cases}$$

6.2.3 Reformulation Guidelines

The following guidelines were used for the functional reformulations.

6.2.3.1 Minimizing Number of Grids

In general, we attempted to create reformulations that resulted in fewer grids and thereby a smaller mixed integer program. However, there are exceptions to this guideline. For example, if we can only choose from uniform grids, then for a function such as $f(x) = \sin(x) + x^2$, it might be useful to have a single grid for both the terms. But if we can use non-uniform grids, then it might be better to choose two different non-uniform grids for the two terms so that we can have an overall better approximation quality.

We now illustrate the effectiveness of using fewer grids for Neumaier-3 and the Shekel Foxholes problems.

a) Shekel Foxholes

The Shekel Foxholes problem, shown in Equation (6.1), can have two different reformulations.

$$\mathrm{Min}\, z = -\sum_{j=1}^{30}\left(\frac{1}{c_j + \sum_{i=1}^{n}(x_i - a_{ji})^2}\right) \qquad s.t.\ 0 \le x_i \le 10,\ i \in \{1,...,n\} \tag{6.1}$$

Reformulation I

$$\mathrm{Min}\, z = -\sum_{j=1}^{30} u_j \tag{6.2}$$

$$u_j = \frac{1}{w_j},\ \ j \in \{1,...,30\} \tag{6.3}$$

$$w_j = c_j + \sum_{i=1}^{n}\left(y_i + (a_{ji})^2 - 2a_{ji}x_i\right),\ j = 1,...,30 \tag{6.4}$$

$$y_i = (x_i)^2,\ i \in \{1,...,n\} \tag{6.5}$$

Reformulation II

$$\mathrm{Min}\, z = -\sum_{j=1}^{30} u_j \tag{6.6}$$

$$u_j = \frac{1}{w_j},\ \ j \in \{1,...,30\} \tag{6.7}$$

$$w_j = c_j + \sum_{i=1}^{n}(v_{ji})^2,\ j = 1,...,30 \tag{6.8}$$

$$v_{ji} = (x_i - a_{ji})^2,\ i \in \{1,...,n\},\ j \in \{1,...,30\} \tag{6.9}$$

Since Reformulation I can provide the same approximation quality with $n+30$ one-dimensional grids as that provided by Reformulation II with $30n+30$ grids, we used Reformulation I for the Shekel and the Shekel Foxholes problems.

b) Neumaier-3

The Neumaier-3 problem, shown in Equation (6.10), can have two different reformulations.

$$\mathrm{Min}\, z = \sum_{i=1}^{n}(x_i - 1)^2 - \sum_{i=2}^{n}(x_i x_{i-1}) \tag{6.10}$$

$s.t. -n^2 \le x_i \le n^2,\ i \in \{1,..,n\}$

<u>Reformulation I</u>

$$\text{Min}\, z = \sum_{i=1}^{n} y_i - \sum_{i=2}^{n} w_i \tag{6.11}$$

$$y_i = (x_i - 1)^2,\ \ i \in \{1,..,n\} \tag{6.12}$$

$$w_i = x_i x_{i-1},\ \ i \in \{2,..,n\} \tag{6.13}$$

<u>Reformulation II</u>

$$\text{Min}\, z = \sum_{i=1}^{n} y_i - \sum_{i=1}^{n/2-1} u_i - u_{n/2} \tag{6.14}$$

$$y_i = (x_i - 1)^2,\ \ i \in \{1,..,n\} \tag{6.15}$$

$$u_i = v_i x_{2i},\ \ i \in \{1,..,n/2 - 1\} \tag{6.16}$$

$$u_{n/2} = x_n x_{n-1} \tag{6.17}$$

$$v_i = x_{2i-1} + x_{2i+1},\ \ i \in \{1,2,..,n/2 - 1\} \tag{6.18}$$

Reformulation I has n one-dimensional grids and $n-1$ two-dimensional grids whereas Reformulation II contains n one-dimensional grids, $n/2-1$ two-dimensional grids and an additional $n/2-1$ constraints. Since Reformulation II leads to a smaller mixed integer program, we used it for the Neumaier-3 problem.

6.2.3.2 Applying Periodic Transformations

For problems that involve trigonometric functions defined over an interval whose length is greater than 2π, we used a trigonometric transformation that significantly improves the approximation quality at the cost of adding a few integer variables to the PLA-based mixed integer program. Consider the function $y = sin(x)$ such that $l \le x \le u,\ u - l \ge 2\pi$. Since this function repeats every 2π units, we can set $x = 2K\pi + w$, where *w* is a continuous variable that lies within [0, 2π] and *K* an integer variable such that $K \in \left[\lceil (l-2\pi)/2\pi \rceil , \lfloor u/2\pi \rfloor \right]$. We obtain these bounds on *K* as follows.

1) Lower bound: $x \geq l$, or, $2K\pi + w \geq l$, or $K \geq (l-w)/2\pi \geq (l-2\pi)/2\pi$
2) Upper bound: $x \leq u$, or, $2K\pi + w \leq u$, or, $K \leq (u-w)/2\pi \leq u/2\pi$

Since *sin*(*x*) = *sin*(*w*), we have effectively defined the approximation over the smaller interval [0, 2π] rather than over the original interval larger [*l*, *u*]. This transformation applies to any periodic function. We used this transformation in problems such as LM-1, LM2-n5, LM2-n10, Shubert, ZeldaSine10 and ZeldaSine20 and could effectively solve them using a coarse 8-segment grid.

6.2.3.3 Scaling Variables

In some problems, the range of one function acts as a domain of another function. Therefore, by reducing the range of values that one function can take, we can reduce the domain of another function and thereby improve the approximation quality of a grid without actually increasing the size of the mixed integer program.

We use scaling to reduce the range of some of the functions in the Storn Tchebychev problem.

$$\text{Min}\, z = p_1 + p_2 + p_3 \tag{6.19}$$

$$u = \sum_{i=1}^{n} (1.2)^{n-i} x_i \tag{6.20}$$

$$p_1 = \begin{cases} (u-d)^2 & if\ u < d \\ 0 & if\ u \geq d \end{cases} \tag{6.21}$$

$$v = \sum_{i=1}^{n} (-1.2)^{n-i} x_i \tag{6.22}$$

$$p_2 = \begin{cases} (v-d)^2 & if\ v < d \\ 0 & if\ v \geq d \end{cases} \tag{6.23}$$

$$w_j = \sum_{i=1}^{n} \left(\frac{2j}{m} - 1 \right)^{n-i} x_i \tag{6.24}$$

$$q_j = \begin{cases} (w_j - 1)^2 & if\ w_j > 1 \\ (w_j + 1)^2 & if\ w_j < -1 \\ 0 & if\ -1 \le w_j \le 1 \end{cases} \tag{6.25}$$

$$p_3 = \sum_{j=0}^{m} q_j \tag{6.26}$$

For $n = 9$: $x_i \in [-256, 256]$, $d = 72.661$, $m = 60$

For $n = 17$: $x_i \in [-32768, 32768]$, $d = 10558.145$, $m = 100$

This problem can be reformulated in two ways: one with the original variables and another with the scaled variables. For Storn Tchebychev-17, the *x* variable can take values from -32768 to 32768. As a result, the *u*, *v* and *w* variables can take values over a very large range, which lead to a very large domain for the square functions. However, if we scale the problem by dividing the *x* variables by 32768, then *x*/32768 takes values between -1 and 1. As a result, the domains for the square functions become very small, and we can get a good approximation even with a coarse 8-segment grid. For example, in the non-scaled problem, *u* lies between -3.47×10^6 and 3.47×10^6 but in the scaled problem, $\bar{u}$ takes values between -17 and 17.

<u>Reformulation I (without scaling)</u>

$$\text{Min}\, z = p_1 + p_2 + p_3 \tag{6.27}$$

$$u = \sum_{i=1}^{n} (1.2)^{n-i} x_i \tag{6.28}$$

$$p_1' \ge d - u \tag{6.29}$$

$$p_1 = (p_1')^2 \tag{6.30}$$

$$v = \sum_{i=1}^{n} (-1.2)^{n-i} x_i \tag{6.31}$$

$$p_2' \geq d - v \tag{6.32}$$

$$p_2 = (p_2')^2 \tag{6.33}$$

$$w_j = \sum_{i=1}^{n} \left(\frac{2j}{m} - 1\right)^{n-i} x_i \quad \text{for } j = 0, \ldots, m \tag{6.34}$$

$$r_j' \geq w_j - 1 \quad \text{for } j = 0, \ldots, m \tag{6.35}$$

$$r_j = (r_j')^2 \quad \text{for } j = 0, \ldots, m \tag{6.36}$$

$$s_j' \geq -w_j - 1 \quad \text{for } j = 0, \ldots, m \tag{6.37}$$

$$s_j = (s_j')^2 \quad \text{for } j = 0, \ldots, m \tag{6.38}$$

$$q_j = r_j + s_j \quad \text{for } j = 0, \ldots, m \tag{6.39}$$

$$p_3 = \sum_{j=0}^{m} q_j \tag{6.40}$$

<u>Reformulation II (with scaling)</u>

$$\text{Min}\, z' = \bar{p}_1 + \bar{p}_2 + \bar{p}_3 \tag{6.41}$$

$$\bar{u} = \sum_{i=1}^{n} (1.2)^{n-i} \left(\frac{x_i}{F}\right) \tag{6.42}$$

$$p_1' \geq \frac{d}{F} - \bar{u} \tag{6.43}$$

$$\bar{p}_1 = (p_1')^2 \tag{6.44}$$

$$\bar{v} = \sum_{i=1}^{n} (-1.2)^{n-i} \left(\frac{x_i}{F}\right) \tag{6.45}$$

$$p_2' \geq \frac{d}{F} - \bar{v} \tag{6.46}$$

$$\bar{p}_2 = (p_2')^2 \tag{6.47}$$

$$\bar{w}_j = \sum_{i=1}^{n} \left(\frac{2j}{m} - 1\right)^{n-i} \left(\frac{x_i}{F}\right) \quad \text{for } j = 0, \ldots, m \tag{6.48}$$

$$r_j' \geq w_j - \frac{1}{F} \quad \text{for } j = 0, \ldots, m \tag{6.49}$$

$$\bar{r}_j = (r_j')^2 \quad \text{for } j = 0, \ldots, m \tag{6.50}$$

$$s_j' \geq -w_j - \frac{1}{F} \quad \text{for } j = 0,\ldots,m \tag{6.51}$$

$$\bar{s}_j = (s_j')^2 \quad \text{for } j = 0,\ldots,m \tag{6.52}$$

$$\bar{q}_j = \bar{r}_j + \bar{s}_j \quad \text{for } j = 0,\ldots,m \tag{6.53}$$

$$\bar{p}_3 = \sum_{j=0}^{m} \bar{q}_j \tag{6.54}$$

6.3 Applying the Basic PLA Method

We now discuss the results that we obtained by applying a uniform 8-segment rectangular grid with a pattern-based model to the various test problems. We call these results *baseline results* since they depict the performance of the basic method without any enhancements in terms of grid design or MIP-modeling.

6.3.1 Baseline Results for Ali et al. Problems

We first discuss the baseline results for the Ali et al. problems which are a set of unconstrained nonlinear problems designed to have multiple localy optimal solutions. 10 of the 55 problems involve functions with discontinuous derivatives whereas the rest involve functions with continuous derivatives. The distribution of the number of variables in the problems is as follows: 24 problems with 2–3 variables, 13 problems with 4–5 variables, 16 problems with 6–10 variables and two problems with 11–20 variables. Further, 25 problems involve trigonometric functions and 13 problems involve the exponential function. For each problem, we used the best reformulation that we could create. Refer to Appendix A7 for more details.

Problem	**NLP Size (cont)**	**MIP Size (cont/bin/constr)**	**MIP nodes**	**Rel. MIP Approx. error| σ**	**Rel.MIP Global Distance δ**	**Rel. MIP Global Obj. Gap μ**	**Rel. Local Global Obj. Gap η**
Ackleys	10	143/96/289	0	0%	0%	0%	0%
AluffiPentini	2	23/16/47	0	100%	0%	100%	0%
BeckerLago	2	23/16/47	0	>100%	0%	0%	0%
Bohachevsky1	2	23/16/47	0	0%	0%	0%	0%
Bohachevsky2	2	103/16/64	0	0%	0%	0%	0%
Branin	2	103/16/64	0	7%	0%	>100%	0%
Camel3	2	109/16/64	0	0%	0%	0%	0%
Camel6	2	103/16/64	0	>100%	0%	>100%	0%
CosMix2	2	23/16/47	0	0%	0%	0%	0%
CosMix4	4	45/32/93	0	0%	0%	0%	0%
Dekkers Aarts	2	103/16/64	0	0%	0%	0%	0%
Easom	2	103/16/64	0	20%	0%	72%	0%
EMichalewicz	**5**	**61/40/121**	**0**	**37%**	**16%**	**47%**	**25%**
Expo	10	122/88/255	0	0%	0%	0%	0%
GoldPrice	2	103/16/64	0	0%	0%	0%	0%
Griewank	10	922/144/579	0	0%	0%	0%	0%
Gulf	3	761/24/95	0	45%	0%	>100%	0%
Hartman3	3	87/56/175	12	21%	88%	34%	0%
Hartman6	6	129/80/253	0	24%	49%	20%	0%
Helical	3	114/24/87	0	>100%	0%	>100%	0%
Hosaki	2	103/16/64	0	8%	0%	5%	0%
Kowalik	4	6603/32/126	0	45%	0%	63%	0%
LM1	3	230/48/176	0	0%	>100%	0%	0%
LM2n5	5	528/88/345	209	27%	14%	>100%	0%
LM2n10	**10**	**1043/168/665**	**5129**	**26%**	**8%**	**>100%**	**>100%**

Table 6.2: Baseline results for Ali et al. problems

Problem	**NLP Size (cont)**	**MIP Size (cont/bin/constr)**	**MIP nodes**	**Rel. MIP Approx. error|** σ	**Rel.MIP Global Distance** δ	**Rel. MIP Global Obj. Gap** μ	**Rel. Local Global Obj. Gap** η
McCormic	2	103/16/64	0	16%	0%	6%	0%
MeyerRoth	**3**	**761/24/95**	**0**	**>100%**	**0%**	**>100%**	**>100%**
MieleCantrell	4	288/32/147	0	0%	0%	0%	0%
ModLangerman	**10**	**206/120/391**	**0**	**7%**	**>100%**	**99%**	**72%**
ModRosenbrock	2	103/16/64	0	100%	0%	>100%	0%
MultiGauss	2	78/56/167	0	100%	>100%	1%	0%
Neumaier2	4	97/64/197	0	25%	24%	>100%	0%
Neumaier3	10	561/112/418	384	48%	>100%	>100%	0%
OddSquare	10	214/116/336	5120	>100%	>100%	>100%	0%
Paviani	10	132/88/265	0	4%	4%	6%	0%
Periodic	2	103/16/64	0	0%	0%	0%	0%
Powell	4	288/32/147	0	0%	0%	0%	0%
Price	**9**	**60682/128/750**	**1207**	**>100%**	**25%**	**>100%**	**>100%**
Rastrigin	10	111/80/231	0	0%	0%	0%	0%
Rosenbrock	10	848/80/401	0	100%	>100%	>100%	0%
Salomon	5	77/56/163	0	0%	0%	0%	0%
Schaffer1	2	103/16/64	0	0%	0%	0%	0%
Schaffer2	2	103/16/64	0	0%	0%	0%	0%
Schwefel	2	111/80/231	0	11%	0%	56%	0%
Shekel5	10	100/72/213	0	6%	>100%	70%	0%
Shekel7	4	122/88/261	0	6%	>100%	69%	0%
Shekel10	4	155/112/333	0	6%	>100%	68%	0%
ShekelFox5	4	386/280/836	28225	12%	>100%	92%	0%
ShekelFox10	5	441/320/951	35002	11%	>100%	96%	0%
Shubert	10	225/96/306	2238	>100%	8%	0.1%	0%

Table 6.2 contd.: Baseline results for Ali et al. problems

Problem	**NLP Size (cont)**	**MIP Size (cont/bin/constr)**	**MIP nodes**	**Rel. MIP Approx. error| σ**	**Rel.MIP Global Distance δ**	**Rel. MIP Global Obj. Gap μ**	**Rel. Local Global Obj. Gap η**
StChebychev9	10	1501/992/3104	0	1%	0%	0%	0%
StChebychev17	20	2469/1632/5104	0	7%	>100%	0%	0%
Wood	9	287/32/146	0	100%	0%	>100%	0%
ZeldaSine10	17	1721/208/915	1199	6%	2%	32%	0%
ZeldaSine20	4	3641/448/1955	8701	6%	4%	40%	0%

Table 6.2 contd.: Baseline results for Ali et al. problems

1. For all except five problems, the basic PLA method is able to provide a starting point that lies in the basin of attraction of the global solution. The problems for which the PLA method does not lead to the global solution are: *EMichalewicz*, *LM2N10*, *MeyerRoth*, *ModLangerman*, and *Price*.
2. For problems such as *LM1*, the periodic function transformation gives a MIP solution that is actually the global solution. This is the not the case when we use a conventional reformulation without the periodic transformation.
3. If we attempt problems such as *LM2n5* and *LM2n10* without the periodic transformation, the quality of the MIP solution, as shown in Table 6.3, is relatively poor. For example, without a periodic transformation, the relative MIP global distance increases from 14% to 25% for *LM2n5* and from 8% to 25% for *LM2n10*.
4. For problems such as StChebychev9 and StChebychev17, scaling the variables leads to a MIP solution that is also the global solution. As shown in Table 6.3, non-scaled reformulations provide MIP solutions that have a very poor quality in terms of the

relative MIP approximation error, which is the difference between the objective value of the MIP solution and the true cost of the MIP solution. This is probably due to the squared terms in the objective function which magnifies the approximation error especially when the range of the variables is large as it is in the non-scaled reformulation.

5. For *oddSquare,* the quality of the MIP solution is not good probably because of the shape of the function in which the contour lines resemble a pond with rectangular ripples.
6. For ShekelFox problems, the function landscape is such that there are a large number of depressions or troughs, one of which is the global minimum. And the MIP solution yielded by a coarse grid happens to be far from the global minimum.

Problem	**NLP Size (cont)**	**MIP Size (cont/bin/constr)**	**MIP nodes**	**Rel.MIP Global Distance δ**	**Rel. MIP Approx. error| σ**	**Abs MIP Global Obj. Gap**	**Abs. Local Global Obj. Gap**
LM2n5	5	381/40/189	0	25.00%	0%	0.1	0.021
LM2n10	10	841/80/394	0	25.00%	0%	0.1469	0
StChebychev9	9	1501/992/3104	0	0.60%	1.43×10^5	0.0009	0
StChebychev17	17	2469/1632/5104	0	29.88%	1.91×10^4	49.8446	0

Table 6.3: Results for poor reformulations of selected Ali et al. problems

7. For problems such as Hartman-3 and Hartman-6, we have a hierarchy of functions such that the range of the functions at a lower level (say Level I) affects the domain of the function at a higher level (say Level II). Consider the Hartman-3 problem given by Equation (6.55).

$$\operatorname{Min} z = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij} \left(x_j - p_{ij} \right)^2 \right) \tag{6.55}$$

s.t. $0 \le x_j \le 1,\ j \in \{1,2,3\}$, constants a_{ij}, p_{ij} and c_i as in Table 1.

i	c_i	a_{ij}			p_{ij}		
		j=1	j=2	j=3	j=1	j=2	j=3
1	1	3	10	30	0.3689	0.117	0.2673
2	1.2	0.1	10	35	0.4699	0.4387	0.747
3	3	3	10	30	0.1091	0.8732	0.5547
4	3.2	0.1	10	35	0.03815	0.5743	0.8828

Table 6.4: Data for Hartman 3

We reformulate this problem as follows.

$$\operatorname{Min} z = -\sum_{i=1}^{4} c_i u_i \tag{6.56}$$

s.t. $$w_i = \sum_{j=1}^{3} y_{ij},\ i \in \{1,2,3,4\} \tag{6.57}$$

$$y_{ij} = a_{ij} \left(x_j - p_{ij} \right)^2,\ i \in \{1,2,3,4\},\ \ j \in \{1,2,3\} \tag{6.58}$$

$$u_i = \exp\left(-w_i\right),\quad i \in \{1,2,3,4\} \tag{6.59}$$

The functions defined by Equations (6.58) are Level I functions whereas the functions defined by Equation (6.59) are Level II functions. We found that in the MIP solution for Hartman-3, the MIP solver always chose a grid point for the Level 1 functions. This phenomenon is also observed in other problems such as Hartman-6 and Odd-Square in which there is a hierarchy of grids and there are no constraints to prevent the dependent variables from taking certain values. The advantage of this phenomeon is that we can need not have any adjacency conditions from the Level-I grids and thereby decrease the size of the overall mixed integer program.

8. The solution time for the *price* problem is quite high (57 seconds) perhaps because of the size of the MIP, which follows from the presence of nine four-dimensional in the problem. We will later discuss how we can solve this problem by using a mix of high and low-dimensional grids such that the low-dimensional grids have a high-resolution and the high dimensional grids have a low resolution.

6.3.2 Baseline Results for Pooling Problems

We now show how the basic PLA method performs on pooling problems which contain the bilinear function in the constraints but have a linear objective function.

Problem	**NLP Size (var/constr)**	**MIP Size (cont/bin/constr)**	**MIP nodes**	**MIP time (CPU s)**	**Rel. MIP Approx. error| σ**	**Rel.MIP Global Distance δ**	**Fraction of feasible constraints φ**	**Rel. Local Global Obj. Gap η**
adhya1	21/40	2789/ 128/ 1022	3954	4.098	6.60%	31.72%	29/40	87.50%
adhya2	25/60	2809/ 128/ 1042	4235	4.748	8.32%	32.99%	45/60	0.00%
adhya3	38/66	2846/ 128/ 1056	4320	7.645	111.65%	36.65%	44/66	18.60%
adhya4	26/48	2802/ 128/ 1038	193	1.159	94.23%	43.48%	40/48	0.00%
foulds2	10/6	1000/ 146/ 686	0	0.061	18.34%	>100%	11/12	0.00%
foulds3	41/28	11888/ 1088/ 5534	0	2.063	56.96%	9.08%	35/40	0.00%
foulds4	26/12	11888/ 1088/ 5534	0	2.198	69.32%	9.37%	33/40	0.00%
foulds5	168/40	5960/ 544/ 2810	55	2.56	75.63%	9.04%	28/36	0.00%
bental4	168/40	259/ 41/ 195	0	0.124	0.00%	>100%	6/6	0.00%
bental5	100/36	2929/ 210/ 1320	0	0.564	100.00%	0.00%	23/28	0.00%
haverly1	9/6	258/ 41/ 193	0	0.156	0.00%	>100%	5/6	0.00%
haverly2	9/6	258/ 41/ 193	0	0.165	>100%	0%	6/6	0%
haverly3	9/6	258/ 41/ 193	0	0.039	>100%	0%	6/6	0%

Table 6.5: Baseline results for pooling problems

- For 9 out of the 11 pooling problems, the MIP solution led CONOPT to obtain the globally optimal solution. The two pooling problems that are currently unsolved are *adhya-1* and *adhya-2*.

6.3.3 Baseline Results for Global-Lib Problems

We now show how the basic PLA method performs on the Global-Lib problems which contain a wide variety of one and higher dimensional nonlinear functions such as the power function, the multiplicative inverse function, the exponential function, the logarithmic function and the bilinear function.

Problem	**NLP Size (cont/ constr)**	**MIP Size (cont/bin/ constr)**	**MIP nodes**	**Rel. MIP Approx. error| σ**	**Rel.MIP Global Distance δ**	**Fraction of feasible constraints φ**	**Rel. Local Global Obj. Gap η**
ex14_1_1	4/5	108/16/72	0	0%	100%	3/5	0%
ex14_1_2	7/10	383/40/199	0	0%	20%	5/10	0%
ex14_1_3	4/5	106/16/70	0	0%	29%	3/5	0%
ex14_1_5	7/7	59102/40/163	0	0%	22%	2/7	0%
ex14_1_6	10/16	336/64/257	0	0%	96%	9/16	0%
ex14_1_8	4/5	105/16/69	0	0%	4%	3/5	0%
ex14_1_9	3/3	14/8/27	0	0%	67%	2/3	0%
ex14_2_1	6/8	6636/48/398	10	0%	0%	5/8	0%
ex14_2_2	5/6	454/32/192	0	0%	0%	4/6	0%
ex14_2_3	7/10	428/72/296	0	0%	3%	6/10	0%
ex14_2_4	6/8	2787/80/435	0	0%	0%	5/8	0%
ex14_2_5	5/6	526/24/188	0	0%	0%	4/6	0%
ex14_2_6	6/8	636/104/429	117	0%	3%	5/8	0%
ex14_2_7	7/10	7394/128/576	0	0%	32%	6/10	0%
ex14_2_8	5/6	511/72/314	0	0%	50%	4/6	0%
ex14_2_9	5/6	690/24/226	23	0%	0%	4/6	0%

Table 6.6: Baseline results for Global-Lib problems

Problem	**NLP Size (cont/ constr)**	**MIP Size (cont/bin/ constr)**	**MIP nodes**	**Rel. MIP Approx. error| σ**	**Rel.MIP Global Distance δ**	**Fraction of feasible constraints φ**	**Rel. Local Global Obj. Gap η**
ex2_1_1	6/2	57/40/118	0	0%	0%	1/2	0%
ex2_1_10	21/11	223/160/473	0	9%	0%	10/11	0%
ex2_1_2	7/3	58/40/119	0	0%	0%	2/3	0%
ex2_1_3	14/10	55/32/103	0	0%	0%	9/10	0%
ex2_1_4	7/6	17/8/29	0	0%	0%	6/6	0%
ex2_1_5	11/12	82/56/174	0	0%	0%	10/12	0%
ex2_1_6	11/6	112/80/237	0	0%	0%	5/6	0%
ex2_1_7	21/11	222/160/472	0	5%	22%	10/11	1%
ex2_1_9	11/2	1257/152/676	10328	12%	37%	1/2	0%
ex3_1_1	9/7	491/64/278	0	0%	27%	4/7	0%
ex3_1_2	6/7	626/40/251	4	>100%	0%	4/7	0%
ex3_1_3	7/7	67/48/145	0	88%	0%	6/7	0%
ex3_1_4	4/4	307/48/196	0	0%	>100%	4/4	0%
ex4_1_1	2/1	12/8/24	0	0%	69%	0/1	0%
ex4_1_3	2/1	12/8/24	0	0%	1%	0/1	0%
ex4_1_4	2/1	12/8/24	0	0%	100%	1/1	0%
ex4_1_6	2/1	12/8/24	0	0%	17%	0/1	0%
ex4_1_7	2/1	12/8/24	0	0%	25%	0/1	0%
ex4_1_8	3/2	23/16/48	0	0%	7%	1/2	0%
ex4_1_9	3/3	15/8/28	0	0%	2%	2/3	0%
ex5_2_2_case1	10/7	201/24/111	0	0%	62%	4/7	0%
ex5_2_2_case2	10/7	201/24/111	0	0%	44%	3/7	0%
ex5_2_2_case3	10/7	201/24/111	0	0%	49%	5/7	0%
ex5_2_4	8/7	567/56/279	0	2%	6%	4/7	0%
ex5_3_2	23/17	1097/80/465	0	0%	7%	8/17	0%

Table 6.6 contd.: Baseline results for Global-Lib problems

Problem	**NLP Size (cont/ constr)**	**MIP Size (cont/bin/ constr)**	**MIP nodes**	**Rel. MIP Approx. error|** σ	**Rel.MIP Global Distance** δ	**Fraction of feasible constraints** φ	**Rel. Local Global Obj. Gap** η
ex5_4_2	9/7	491/64/278	0	0%	36%	3/7	0%
ex5_4_3	17/14	781/96/430	0	0%	0%	13/14	0%
ex6_1_2	5/4	371/32/170	0	>100%	79%	1/4	>100%
ex6_1_4	7/5	4444/48/308	21	>100%	55%	1/5	0%
ex6_2_14	5/3	1029/32/323	0	0%	0%	2/3	0%
ex7_2_2	7/6	391/48/216	0	0%	>100%	1/6	0%
ex8_3_1	116/113	16969/840/ 6053	14060	0%	89%	11/78	0%

Table 6.6 contd.: Baseline results for Global-Lib problems

- 46 out of 48 problems could be solved to global optimality using the PLA approach. In Section 6.8, we discuss the performance of CONOPT on the Ali et al. and the Global-Lib problems using a default starting point in which every variable has a value of zero.
- *ex2_1_7* and *ex8_3_1*, which could not be solved to global optimality, are solved using a finer grid. Refer to Section 6.4.2 for more details.
- For all problems except *ex14_1_5*, *ex14_2_1*, and *ex2_1_9*, the MIP was solved within 1 second. For these three problems, the Cplex took between two to three seconds to solve the PLA-based MIP.
- For most of the problems, the relative MIP approximation error is zero. This implies that the MIP solution occurs at a gid point in spite of the presence of constraints.

6.3.4 Baseline Results for MINLP-Lib Problems

We now discuss the results when we apply the PLA method to mixed integer nonlinear problems which are the hardest problems in our set. As part of the solution process, we fix the integer variables in the MINLP to their PLA solution values and then use CONOPT to solve the resulting continuous nonlinear program.

Problem	**NLP Size (cont/int/constr)**	**MIP Size (cont/bin / constr)**	**MIP nodes**	**MIP time (CPU sec)**	**Rel.MIP Global Distance δ**	**Fraction of feasible constraints φ**	**Fraction of correctly identified integer variables ψ**	**Rel. Local Global Obj. Gap η**
ex1221	3/3/6	**19/52**	**0**	**0.0**	**61%**	3/6	**2/3**	**3%**
ex1222	3/1/4	9/28	0	0.0	0%	3/4	1/1	0%
ex1223	8/4/14	**60/180**	**0**	**0.1**	**47%**	13/14	**3/4**	**27%**
ex1223a	4/4/10	**28/82**	**0**	**0.0**	**42%**	9/10	**3/4**	**27%**
ex1223b	4/4/10	**60/176**	**0**	**0.1**	**42%**	9/10	**3/4**	**27%**
ex1224	4/8/8	32/105	0	0.1	3%	5/8	8/8	0%
ex1225	3/6/11	22/74	0	0.0	0%	11/11	6/6	0%
ex1226	3/3/6	19/71	0	0.1	0%	5/6	3/3	0%
ex1233	41/12/65	236/981	0	1.7	12%	56/65	12/12	0%
ex1244	73/23/130	397/1705	86	2.6	59%	122/130	23/23	0%
ex1252	25/15/44	**255/1112**	**9570**	**5.6**	**15%**	35/44	**10/15**	**2%**
ex1252a	16/9/35	**243/1103**	**8325**	**5.1**	**93%**	28/35	**2/9**	**9%**
ex1263	21/72/56	232/800	358	0.6	26%	56/56	62/72	0%
ex1263a	1/24/36	**164/780**	**2579**	**1.7**	**65%**	34/36	**6/24**	**9%**
ex1264	21/68/56	228/800	734	1.4	54%	55/56	57/68	0%
ex1264a	1/24/36	**164/780**	**676**	**0.8**	**222%**	32/36	**5/24**	**3%**

Table 6.7: Baseline results for MINLP-Lib problems

Problem	**NLP Size (cont/int/constr)**	**MIP Size (cont/bin / constr)**	**MIP nodes**	**MIP time (CPU sec)**	**Rel.MIP Global Distance δ**	**Fraction of feasible constraints φ**	**Fraction of correctly identified integer variables ψ**	**Rel. Local Global Obj. Gap η**
ex1265	31/100/75	340/1210	558	1.2	0%	74/75	100/100	0%
ex1265a	1/35/45	245/1180	9287	6.7	266%	44/45	12/35	0%
ex1266	43/138/96	474/1704	0	0.5	35%	96/96	131/138	0%
ex1266a	1/48/54	342/1662	3485	6.1	314%	54/54	11/48	0%
gasnet	81/10/70	**598/2816**	**15323**	**601.5**	**11%**	32/70	**5/10**	**2%**
synheat	45/12/65	236/984	0	1.7	10%	61/65	12/12	0%

Table 6.7 contd.: Baseline results for MINLP-Lib problems

1. For 13 out of 22 problems, the PLA approach could lead to the globally optimal solution.
2. The problems for which the local solution was not the global solution (i.e., *ex1221*, *ex1223*, *ex1223a*, *ex1223b*, *ex1252*, *ex1252a*, *ex1263a*, *ex1264a*, and *gasnet*) become candidates for applying more sophisticated techniques.

3. In Section 6.4.3, we discuss how to solve *ex1221*, *ex1223*, *ex1223a*, *ex1223b*, *ex1252*, and *ex1252a* using a finer grid. In Section 6.11, we discuss how to solve *gasnet*. The PLA method could not solve *ex1263a* and *ex1264a*.

6.4 EFFECT OF INCREASED GRID RESOLUTION

For those problems for which the MIP solution quality as measured by the MIP-global objective difference was high, we applied the PLA method with high-resolution grids and found that for many problems, the solution quality improves significantly as the grid becomes finer.

6.4.1 Effect of Increased Grid Resolution on Ali et al. Problems

We first discuss the effect of applying high resolution grids to the Ali et al. problems. Here we use another metric which we call the *absolute MIP-Global Objective gap* which is the absolute difference between the objective value of the global solution and the true cost of the MIP solution. This metric is more useful than the *relative MIP-Global Objective gap* for problems that have a global objective value of zero since it helps us observe how for these problems, the true cost of the MIP solution approaches the objective value of the global solution as we increase the grid resolution.

Problem	No. of Segments	MIP nodes	Rel.MIPGlobal Distance δ	Rel. MIP Approx. error| σ	Abs MIP-Global Obj. gap	Rel. MIP Global Obj. Gap μ
AluffiPentini	32	0	19.42%	0.00%	0.0565	16.03%
AluffiPentini	64	0	10.43%	0.00%	0.0123	3.49%
AluffiPentini	128	0	4.50%	0.00%	0.0026	0.75%
Camel6	32	0	48.24%	0.00%	0.5155	49.97%
Camel6	64	0	2.10%	0.00%	0.0012	0.11%
Camel6	128	0	2.10%	0.00%	0.0012	0.11%
EMichalewicz	32	0	28.53%	0.00%	0.4705	10.04%
EMichalewicz	64	0	19.84%	0.00%	0.1749	3.73%
EMichalewicz	128	0	0.53%	0.00%	0.0402	0.86%
Gulf	32	0	16.84%	0.00%	0.0023	100%
Gulf	64	0	5.50%	0.00%	0.0006	100%
Helical	32	0	25.00%	0.00%	6.25	100%
Helical	64	0	6.25%	0.00%	0.3906	100%
Helical	128	0	6.25%	0.00%	0.3906	100%

Table 6.8: Effect of increased grid resolution on Ali et al. problems

Problem	**No. of Segments**	**MIP nodes**	**Rel.MIPGlobal Distance** $\boldsymbol{\delta}$	**Rel. MIP Approx. error|** $\boldsymbol{\sigma}$	**Abs MIP-Global Obj. gap**	**Rel. MIP Global Obj. Gap** $\boldsymbol{\mu}$
MeyerRoth	32	0	9.74%	0.00%	0.0001	125.64%
MeyerRoth	64	0	4.20%	0.00%	0	42.85%
ModLangerman	32	0	1.55%	27.20%	0.2096	21.72%
ModLangerman	64	0	0.54%	2.94%	0.0285	2.96%
ModRosenbrock	32	0	6.25%	0.00%	0.441	100%
ModRosenbrock	64	0	128.71%	0.00%	0.1006	100%
ModRosenbrock	128	0	122.72%	0.00%	0.0103	100%
Neumaier2	32	0	20.34%	4.79%	12.1327	100%
Neumaier2	64	0	3.59%	4.20%	4.6847	100%
Neumaier2	128	0	0.42%	2.29%	0.4459	100%
Neumaier3	32	4072	9.27%	21.63%	29.375	13.99%
Neumaier3	64	17188	3.06%	3.60%	6.4258	3.06%
Price	6–16	1367	114.37%	18.33%	136264	0%
Rosenbrock	32	0	100.00%	800.00%	1	100%
Rosenbrock	64	0	6.25%	9.00%	0.3472	100%
Rosenbrock	128	0	6.25%	9.00%	0.3472	100%
Shekel5	32	0	1.56%	14.99%	1.3469	13.27%
Shekel5	64	0	1.56%	14.60%	1.3469	13.27%
Shekel5	128	363	0.39%	0.88%	0.0978	0.96%
Shekel7	32	0	7.95%	335.86%	7.8025	75.00%
Shekel7	64	253	1.78%	18.80%	1.6815	16.16%
Shekel7	128	0	0.39%	0.90%	0.098	0.94%
Shekel10	32	194	7.95%	319.67%	7.8033	74.06%
Shekel10	64	1001	1.78%	18.52%	1.6802	15.95%
Shekel10	128	2041	0.39%	0.89%	0.0985	0.93%

Table 6.8 contd.: Effect of increased grid resolution on Ali et al. problems

Problem	**No. of Segments**	**MIP nodes**	**Rel.MIPGlobal Distance δ**	**Rel. MIP Approx. error| σ**	**Abs MIP-Global Obj. gap**	**Rel. MIP Global Obj. Gap μ**
Wood	32	0	84.09%	0.00%	10.3594	100%
Wood	64	0	6.25%	0.00%	0.8164	100%
Wood	128	0	25.48%	0.00%	0.533	100%
ZeldaSine10	16	182	3.00%	1.22%	0.4432	12.66%
ZeldaSine20	16	8896	3.13%	1.00%	0.7689	21.97%

Table 6.8 contd.: Effect of increased grid resolution on Ali et al. problems

1) For problems such as AluffiPentini, Camel6, EMichalewicz, Neumaier3, Shekel5, Shekel7, and Shekel10, 128-segment grids can provide MIP solutions that have a cost that is within 1% of the objective value of the global solution.

2) Out of the five unsolved problems, two problems (MeyerRoth, and ModLangerman) can be solved to global optimality by using finer grids.

3) If a problem contains both low-dimensional and high-dimensional functions, then we can use a high-resolution grid for the low-dimensional functions and low-resolution grids for the high-dimensional functions. We employ this method in solving the price problem, which is expressed as follows

$$\mathrm{Min}\, z = \gamma^2 + \sum_{k=1}^{4} \left(\alpha_k^2 + \beta_k^2\right) \tag{6.60}$$

$$\alpha_k = (1 - x_1 x_2) x_3 \{\exp[x_5 (g_{1k} - g_{3k} * 0.001 * x_7 - g_{5k} * 0.001 * x_8)] - 1\} - g_{5k} + g_{4k} x_2$$

$$\text{for } k = 1,\ldots,4 \tag{6.61}$$

$$\beta_k = (1 - x_1 x_2) x_4 \{\exp[x_6 (g_{1k} - g_{2k} - g_{3k} * 0.001 * x_7 + g_{4k} * 0.001 * x_9)] - 1\} - g_{5k} x_1 + g_{4k}$$

$$\text{for } k = 1,\ldots,4 \qquad (6.62)$$

$$\gamma = x_1 x_3 - x_2 x_4 \qquad (6.63)$$

Reformulation

$$\text{Min}\, z = t + \sum_{k=1}^{4} \left(p_k + q_k \right) \qquad (6.64)$$

$$u = (1 - x_1 x_2) x_3 \qquad (6.65)$$

$$v = (1 - x_1 x_2) x_4 \qquad (6.66)$$

$$r_k = g_{1k} - g_{3k} * 0.001 * x_7 - g_{5k} * 0.001 * x_8 \text{ for } k = 1,\ldots,4 \qquad (6.67)$$

$$s_k = g_{1k} - g_{2k} - g_{3k} * 0.001 * x_7 + g_{4k} * 0.001 * x_9 \text{ for } k = 1,\ldots,4 \qquad (6.68)$$

$$p_k = \left(u\{\exp[x_5 r_k] - 1\} - g_{5k} + g_{4k} x_2 \right)^2 \text{ for } k = 1,\ldots,4 \qquad (6.69)$$

$$q_k = \left(v\{\exp[x_6 s_k] - 1\} - g_{5k} x_1 + g_{4k} \right)^2 \text{ for } k = 1,\ldots,4 \qquad (6.70)$$

$$t = (x_1 x_3 - x_2 x_4)^2 \qquad (6.71)$$

We used three-dimensional grids for Equations (6.65) and (6.67), and four-dimensional grids for linearizing Equations (6.68 – 6.71). Further, by using 6 segments per dimension for the four-dimensional grids and 16-segments per dimension for the three-dimensional grids, we are able to get a MIP solution that lies in the basin of attraction of the global solution.

4) In Section 6.5, we discuss how we solved the two unsolved Ali et al. problems (EMichalewicz and LM2n10) using non-uniform grids.

6.4.2 Effect of Increased Grid Resolution on Global-Lib Problems

We now discuss the effect of applying high resolution grids to Global-Lib problems.

Problem	No. of Segments	MIP Size (cont/bin/ constr)	MIP time (CPU sec)	MIP nodes	Rel.MIP Global Distance δ	Fraction of feasible constraints φ	Rel. Local Global Obj. Gap η
ex14_1_3	8	106/16/70	0.031	0	29.37%	3/5	0.00%
ex14_1_3	16	330/32/134	0.037	0	11.78%	3/5	0.00%
ex14_2_7	8	7394/128/576	0.655	0	32.14%	6/10	0.00%
ex14_2_7	16	86146/256/11 20	281.3 6	426	0.45%	6/10	0.00%
ex14_2_8	8	511/72/314	0.047	0	50.32%	4/6	0.00%
ex14_2_8	16	1623/144/610	0.234	4	1.71%	4/6	0.00%
ex2_1_7	8	222/160/472	0.179	0	21.70%	10/11	0.76%
ex2_1_7	16	382/320/952	0.125	0	0.00%	7/11	0.00%
ex2_1_9	8	1257/152/676	2.447	10328	37.34%	1/2	0.00%
ex2_1_9	16	4113/304/134 0	9.227	21272	14.10%	1/2	0.00%
ex3_1_1	8	491/64/278	0.108	0	26.84%	4/7	0.00%
ex3_1_1	16	1595/128/550	0.468	618	4.13%	4/7	0.00%
ex4_1_1	8	12/8/24	0.001	0	68.52%	0/1	0.00%
ex4_1_1	16	20/16/48	0	0	0.32%	0/1	0.00%
ex4_1_6	8	12/8/24	0.001	0	16.67%	0/1	0.00%
ex4_1_6	16	20/16/48	0	0	4.17%	0/1	0.00%
ex4_1_7	8	12/8/24	0.003	0	25.00%	0/1	0.00%
ex4_1_7	16	20/16/48	0	0	25.00%	0/1	0.00%
ex5_2_2_case1	8	201/24/111	0.007	0	62.44%	4/7	0.00%
ex5_2_2_case1	16	641/48/215	0.079	0	43.60%	3/7	0.00%

Table 6.9: Effect of increased grid resolution on Global-Lib problems

Problem	No. of Segments	MIP Size (cont/bin/ constr)	MIP time (CPU sec)	MIP nodes	Rel.MIP Global Distance δ	Fraction of feasible constraints φ	Rel. Local Global Obj. Gap η
ex5_2_2_case2	8	201/24/111	0.149	0	44.41%	3/7	0.00%
ex5_2_2_case2	16	641/48/215	0.031	0	24.35%	3/7	0.00%
ex5_2_2_case3	8	201/24/111	0.009	0	49.12%	5/7	0.00%
ex5_2_2_case3	16	641/48/215	0.25	0	32.57%	4/7	0.00%
ex5_4_2	8	491/64/278	0.188	0	35.83%	3/7	0.00%
ex5_4_2	16	1595/128/550	0.406	485	6.18%	4/7	0.00%
ex6_1_2	8	371/32/170	0.007	0	79.16%	1/4	100.01%
ex6_1_2	16	1235/64/330	0.328	61	20.02%	1/4	0.00%
ex6_1_4	8	4444/48/308	0.313	21	55.18%	1/5	0.00%
ex6_1_4	16	29596/96/596	2.826	101	0.96%	1/5	0.00%

Table 6.9 contd.: Effect of increased grid resolution on Global-Lib problems

1) The two Global-Lib problems (ex2_1_7 and ex6_1_2) which could not be solved using the basic PLA method could be solved to global optimality by increasing the grid resolution.
2) The relative MIP-Global distance metric also improves as we increase the grid granularity. However, in most cases, the fraction of feasible constraints does not improve with grid resolution.

6.4.3 Effect of Increased Grid Resolution on MINLP-Lib Problems

We now discuss the effect of applying the PLA method with high resolution grids to MINLP-Lib problems.

Problem	**No. of Segments**	**MIP Size (cont/bin/constr)**	**MIP nodes**	**MIP time (CPU sec)**	**Rel.MIP Global Distance δ**	**Fraction of feasible constraints φ**	**Rel. Local Global Obj. Gap η**
ex1221	8	23/19/52	0	0.0	61%	3/6	3%
	16	39/35/100	0	0.0	2%	4/6	0%
ex1223	8	83/60/180	0	0.1	47%	13/14	27%
	16	139/116/348	0	0.0	1%	13/14	0%
ex1223a	8	37/28/82	0	0.0	42%	9/10	27%
	16	61/52/154	0	0.0	2%	9/10	0%
ex1223b	8	79/60/176	0	0.1	42%	9/10	27%
	16	135/116/344	0	0.0	2%	9/10	0%
ex1252	8	21223/255/1112	9570	5.6	15%	35/44	2%
	16	255463/495/2168	10058	17.4	13%	33/44	0%
ex1252a	8	21220/243/1103	8325	5.1	93%	28/35	9%
	16	255460/483/2159	21220	31.0	69%	25/35	0%
ex1263a	8	1513/164/780	2579	1.7	65%	34/36	9%
	16	**5001/324/1516**	**8940**	**33.7**	**60%**	**36/36**	**2%**
ex1264a	8	1513/164/780	676	0.8	222%	32/36	3%
	16	**5001/324/1516**	**1271**	**4.0**	**188%**	**32/36**	**3%**

Table 6.10: Effect of increased grid resolution on MINLP-Lib problems

1) Of the nine MINLP problems that could not be solved by uniform 8-segment grids, six could be solved by increasing the grid resolution.

2) In many cases, the relative MIP-global distance improves as the grid becomes refined. However, the fraction of feasible constraints generally remains the same.
3) We do not show *gasnet* in the table above since with a 16-segment grid, Cplex could not find any feasible MIP solution within 600 seconds. In Section 6.11, we show how to solve *gasnet* using a combination of problem reduction technqiues and constraint-based valid inequalities.
4) Two problems (*ex1263a* and *ex1264a*) which could not to be solved to global optimality contain product functions of pure (non-binary) integers. These problems are the only MINLP problems that we could not solve using the PLA approach. *adhya1* and *adhya3* are the other two problems that we could not solve using the PLA approach.

6.5 Effect of Non-uniform Grids

Some problems for which we need high-resolution uniform grids to get good quality MIP solutions can be solved quite effectively by low resolution non-uniform grids. These include Ali et al. problems *Epistatic Michalewicz*, LM2n5 and LM2n10, and Global-Lib problem *ex_8_3_1*. The Ali et al. problems involve trigonometric functions, whereas the *ex_8_3_1* involves a two-dimensional exponential function used in the design of chemical reactor design. For *Epistatic Michalewicz*, we require uniform one-dimensional grids with 128 segments before we can get an MIP solution that lies in the basin of attraction of the global solution. However, we can solve this problem with only 8 segments if we use a non-uniform grid that creates these 8 segments by aggregating 128 intervals. As shown in Equation (6.72), the nonlinear program for this problem contains the function $f(y_1) = \sin(y_1)(\sin(y_1^2/\pi))^{20}$. The shape of this function is shown in Figure

6.1, which also shows the position of the break points obtained from the shortest path-based method. As we can see in the figure, the grid resolution is finer only in those regions where the curvature of the function is high, which results in good approximation quality with fewer segments.

$$\operatorname{Min} z = -\sum_{i=1}^{n} \sin(y_i)(\sin(i y_i^2/\pi))^{2m} \text{where } y_i = \begin{cases} x_i\cos(\theta) - x_{i+1}\sin(\theta),\ i = 1,3,..,< n \\ x_i\cos(\theta) + x_{i-1}\sin(\theta),\ i = 2,4,...,< n \\ x_i,\ i = n \end{cases}$$

$$0 \le x_i \le \pi,\ i \in \{1,\ 2,...,n\},\ \theta = \pi/6,\ m = 10 \qquad (6.72)$$

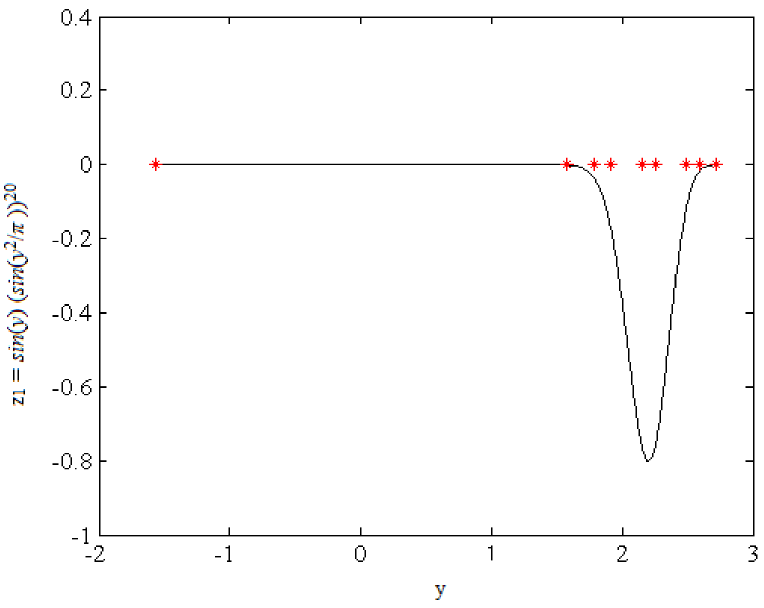


Figure 6.1: Non-uniform segments in Epistatic Michalewicz

Similarly, for LM2n10, we need a rectangular grid with 12 segments along each dimension before we can get into the basin of attraction of the global solution. If we use a Crisscross grid with the same resolution (144 cells created by 6 segments per dimension), then the absolute difference between the true cost of the MIP solution and the objective value of the global solution decreases from 0.0485 to 0.0427. However, if we use a non-uniform Crisscross grid in which we aggregate 36 intervals over 6 segments, we further decrease this difference to 0.0064. Thus, the MIP solution quality improves as we use a carefully designed non-uniform grid. The same behavior is exhibited by LM2n5.

The Global-Lib problem *ex8_3_1* deals with designing a chemical reactor network to maximize the yield of a specific chemical product. The complete nonlinear formulation and the PLA-based reformulation are described in Appendix A10. The nonlinear program involves two types of nonlinear functions: bilinear functions and non-linear functions of the form $f(x,y) = x\exp(-b/y)$ where b is a constant. For b = 7971, we graphically depict this function and its contour lines in Figure 6.2. If we try to solve this problem using 2x2 (i.e., two segments along the two axes) uniform rectangular grids for linearizing the bilinear functions and 4x4 uniform Crisscross grids for the exponential functions, then we cannot find a MIP solution that lies in the basin of attraction of the global solution. However, if instead we use 2x2 uniform rectangular grids for linearizing the bilinear functions and 4x4 non-uniform Crisscross grids (based on the alternating shortest-path method) for the exponential functions, then we can obtain a MIP solution that leads to a local solution whose objective value is within 3.46% of the global objective value. We show the non-uniform grid in Figure 6.3. As we can see, the alternating shortest path method results in a grid that has a higher resolution in those areas of the domain where the curvature of the function is high.

Problem	Grid Type	No. of Segments/No. of Intervals	Rel.MIP Global Distance δ	Rel. MIP Approx. error| σ	Abs MIP-Global Obj. Gap	Rel. Local Global Obj. Gap η
EMichalewicz	Uniform/1D	8	36.86%	15.89%	2.1821	25.43%
EMichalewicz	Non-uniform/ID	8/128	1.28%	0.00%	0.0746	0%
LM2n5	Uniform/Rec	12	14.65%	105.02%	0.0208	0%
LM2n5	Uniform/Rec	6	13.15%	258.82%	0.0149	0%
LM2n5	Non-uniform/CC	6/36	4.98%	194.01%	0.0045	0%
LM2n10	Uniform/Rec	12	15.69%	44.92%	0.0485	0%
LM2n10	Uniform/Rec	6	15.01%	90.47%	0.0427	0%
LM2n10	Non-uniform/CC	6/36	5.24%	292.79%	0.0064	0%
ex8_3_1	Uniform/Rec/CC	Rec: 2/2, CC: 4/4	115%	0%	-	>100%
ex8_3_1	Non-uniform/Rec/CC	Rec: 2/2, CC: 4/8	99.6%	0%	-	3.5%

Table 6.11: Effect of using non-uniform grids

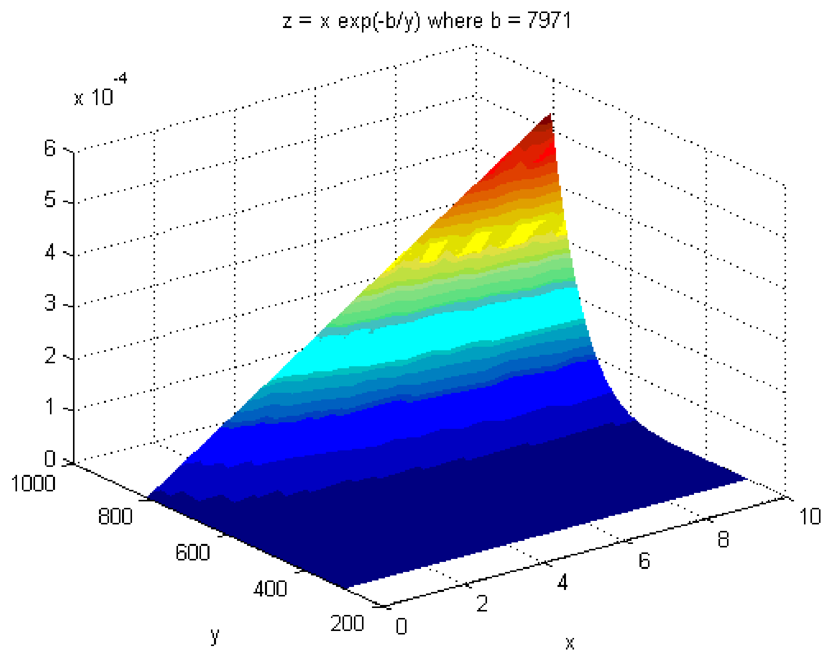


Figure 6.2: Plot of two-dimensional exponential function in *ex8_3_1*

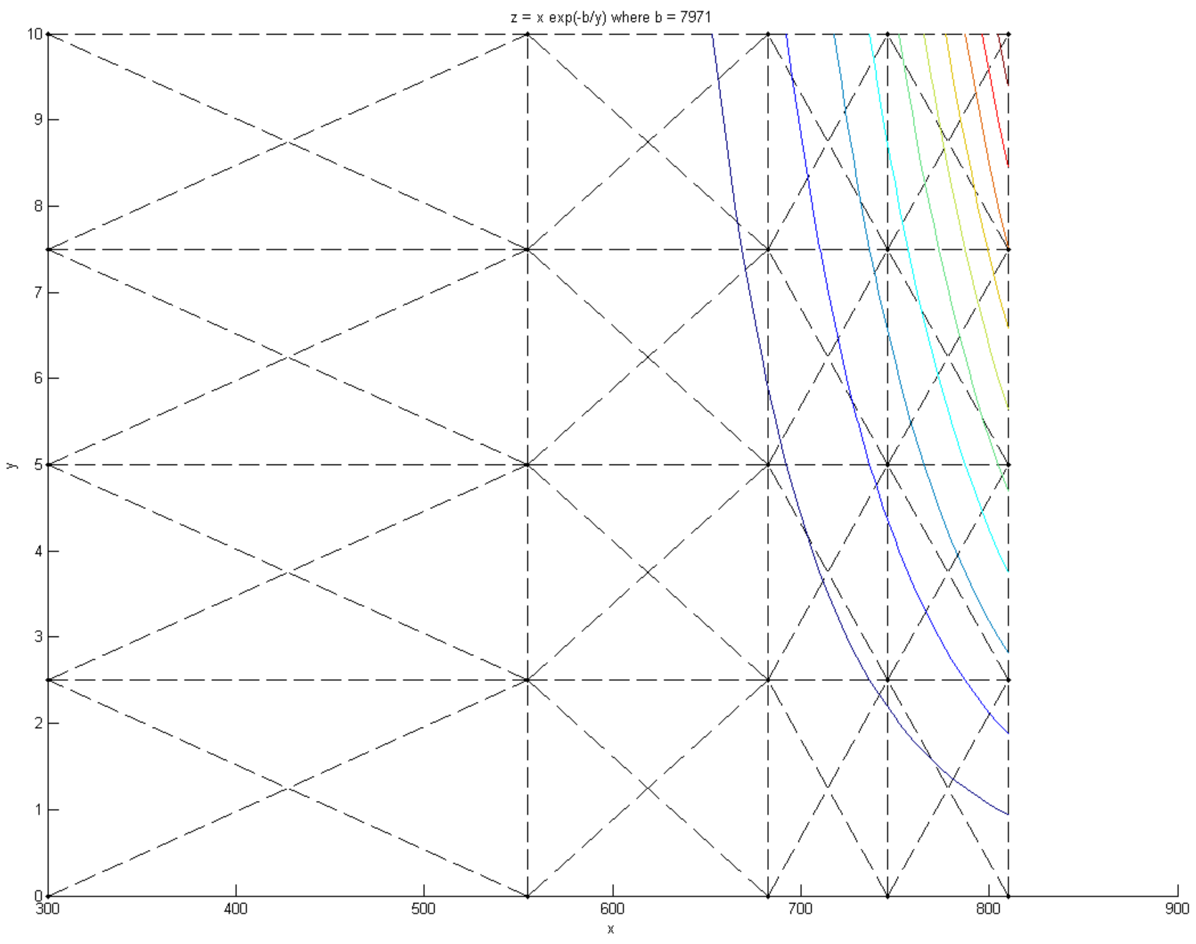


Figure 6.3: Nonuniform grid for the exponential function in *ex8_3_1*

The problem of obtaining the best non-uniform grid for a two-dimensional function can be formulated as a set-covering problem as described in Appendix A8. However, Cplex takes a long time to solve this pure-integer program even for medium grid sizes. For certain functions, we have solved this set-covering problem for low grid resolutions, obtained the best non-uniform grids and then compared the approximation quality of the best non-uniform grid with that of the one obtained from the shortest-path-based method. We report the results in Table 6.12. To estimate the overall approximation error, we use a set of sample points located at equidistant points along the domain of the function and compute the sum of the absolute errors over all the sample points. As is clear from the data, the approximation error from the shortest-path-based grid is the same as that of the grid obtained from the set covering problem. Thus, for these specific grid resolutions and these specific functions, the shortest-path gives the best non-uniform grid.

Function	**Domain**	**No. of Segments/No of Intervals**	**No. of Sample Points**	**Approx. Error using Set Covering Formulation**	**Approx. Error using Alternating Shortest Path-based Method**
$y_1 = x_1 x_2$	$1 \leq x_1 \leq 2$ $10 \leq x_2 \leq 20$	6/18	4000	114459.9	114459.9
$y_2 = x_1 (x_2)^2$	$1 \leq x_1 \leq 2$ $10 \leq x_2 \leq 20$	6/18	4000	3640467.8	3640467.8
$y_3 = (x_1)^2 x_2$	$1 \leq x_1 \leq 2$ $10 \leq x_2 \leq 20$	6/18	4000	365905.1	365905.1

Table 6.12: Benchmarking grids generated by the shortest-path-based method

6.6 Effect of Grid Shape

To understand the effect of grid shape, we compare the performance of 16-segment rectangular grids with 8-segment Crisscross grids (both with 256 cells) for selected pooling problems. We used the pattern-based model with cumulative adjacency for these runs

Problem	Grid Type	MIP Size (cont/bin/constr)	Rel.MIP Global Distance δ	Rel. Local Global Obj. Gap η
adhya1	Rectangular	(9573/256/1918)	2.74%	87.50%
	Crisscross	(4837/192/1150)	0.48%	87.50%
adhya2	Rectangular	(9573/256/1938)	6.00%	88.18%
	Crisscross	(4857/192/1170)	5.41%	0.00%
adhya3	Rectangular	(9630/256/1952)	112.43%	88.41%
	Crisscross	(4894/192/1184)	112.45%	0.00%
adhya4	Rectangular	(9586/256/1934)	93.04%	0.00%
	Crisscross	(4850/192/1166)	93.04%	0.00%

Table 6.13: Effect of using different grid shapes

1. The results indicate that in general, the Crisscross grid does better than the rectangular grid either in terms of the relative MIP global distance or in terms of the relative NLP-global gap, which indicates the gap between the objective value of the local solution and that of the global solution.
2. These results are expected because in a Crisscross grid, each point in the function domain had a unique representation in terms of the vertices of the cell in which it lies. This is not true for a rectangular grid in which a point can have multiple convex combination representations, some of which can have inferior approximation quality. However, because of this non-unique representation, we can solve the MIP (for pooling problems) using rectangular grids in less time than by using Crisscross grids.

6.7 Effect of Indexing Scheme

To understand the impact of the indexing scheme on the performance of the PLA approach, we apply the PLA method to selected pooling propbelms using rectangular grids with (a) pattern based cumulative adjacency model and (b) pattern-based logarithmic model. A time limit of 600 seconds was imposed on the MIP solver.l

Problem	No. of segments	Non-logarithmic Indexing			Logarithmic Indexing		
		MIP Size (cont/bin/constr)	MIP Time	MIP Gap	MIP Size (cont/bin/constr)	MIP Time	MIP Gap
adhya1	16	9573/256/1918	58.8	0.00%	9573/64/1342	56.5	0.00%
adhya2	16	9593/256/1938	31.3	0.00%	9593/64/1362	58.8	0.00%
adhya3	16	9630/256/1952	91.2	0.00%	9630/64/1376	57.0	0.00%
adhya4	16	9586/256/1934	12.0	0.00%	9586/64/1358	6.1	0.00%
adhya1	32	35429/512/3710	601.6	1.98%	35429/80/2398	600.1	2.38%
adhya2	32	35449/512/3730	600.3	1.92%	35449/80/2418	600.1	7.81%
adhya3	32	35486/512/3744	600.1	0.36%	35486/80/2432	600.1	0.95%
adhya4	32	35442/512/3726	60.4	0.00%	35442/80/2414	430.8	0.00%

Table 6.14: Effect of using different indexing schemes

1. The results indicate that, in general, for medium grid resolutions (that we have considered), the logarithmic indexing scheme does not offer a significant advantage over the non-logarithmic indexing scheme in terms of MIP times and MIP gaps at termination.

2. These results are expected because advantages of a logarithmic indexing scheme become more pronounced only at higher grid resolutions. However, for most of our problems, even medium granularity grids yield good MIP solutions that lie in the basin of attraction of the global solution.

6.8 Comparison With CONOPT

We now compare the performance of the PLA approach with that of CONOPT using the default CONOPT settings and without a specific starting point. We use the Ali et al. problems, the Global-Lib problems and the pooling problems for this comparison. As shown in Table 6.15 and Table 6.16, CONOPT could not solve 21 Ali et al. problems and 20 Global-Lib problems. However, we can solve all these problems using the PLA approach. Further, only 3 of the 13 pooling problems could be solved using CONOPT whereas with the PLA method, we could solve 11 of the 13 pooling problems.

Problem	**Global Objective Value**	**Objective Value CONOPT**	**Absolute Gap CONOPT**	**Relative Gap CONOPT**
beckerLago	0.00	50.00	50.00	100%
camel6	-1.03	0.00	1.03	100%
dekkersAarts	-24776.52	0.00	24776.52	100%
easom	-1.00	0.00	1.00	100%
eMichalewicz	-4.69	0.00	4.69	100%
goldPrice	3.00	30.00	27.00	900%
hosaki	-2.35	0.00	2.35	100%
lm1	0.00	1.12	1.12	100%
meyerRoth	0.00	0.12	0.12	100%
modLangerman	-0.97	0.00	0.97	100%
oddSquare	-1.14	-1.00	0.14	13%
price	0.00	190.80	190.80	100%
schwefel	-4189.83	0.00	4189.83	100%
shekel5	-10.15	-5.06	5.10	50%
shekel7	-10.40	-5.09	5.32	51%
shekel10	-10.54	-5.13	5.41	51%
shekelFox5	-10.41	-1.58	8.83	85%
shekelFox10	-10.21	-1.48	8.73	86%
shubert	-186.73	0.07	186.80	100%
zeldaSine10	-3.50	-1.00	2.50	71%
zeldaSine20	-3.50	-1.00	2.50	71%

Table 6.15: Ali et al. Problems that could not be solved by CONOPT

Problem	**Global Objective Value**	**Objective Value CONOPT**	**Absolute Gap CONOPT**	**Relative Gap CONOPT**
ex14_1_6	0.00	1.00	1.00	100%
ex14_1_8	0.00	0.04	0.04	100%
ex2_1_1	-17.00	0.00	17.00	100%
ex2_1_3	-15.00	-10.11	4.89	33%
ex2_1_6	-39.00	-21.13	17.88	46%
ex2_1_7	-4150.41	-407.25	3743.16	90%
ex2_1_9	-0.38	-0.33	0.04	11%
ex3_1_3	-310.00	-132.00	178.00	57%
ex4_1_1	-7.49	-0.52	6.97	93%
ex4_1_3	-443.67	0.00	443.67	100%
ex4_1_6	7.00	250.00	243.00	3471%
ex4_1_9	-5.51	-4.05	1.45	26%
ex5_2_2_case1	-400.00	0.00	400.00	100%
ex5_2_2_case2	-600.00	0.00	600.00	100%
ex5_2_2_case3	-750.00	0.00	750.00	100%
ex5_2_4	-450.00	0.00	450.00	100%
ex5_3_2	1.86	1.00	-0.87	46%
ex5_4_3	4845.46	5801.02	955.56	20%
ex6_1_4	-0.29	0.00	0.29	100%
ex6_2_14	-0.70	0.10	0.79	114%

Table 6.16: Global-Lib problems that could not be solved by CONOPT

Problem	Global Objective Value	Objective Value CONOPT	Relative Gap CONOPT	Relative Gap PLA
adhya1	-549.8031	0	100.0%	87.50%
adhya2	-549.8031	0	100.0%	0.00%
adhya3	-561.0447	0	100.0%	18.60%
adhya4	-877.6457	0	100.0%	0.00%
foulds2	-1100	-1000	9.1%	0.00%
foulds3	-8	-7	12.5%	0.00%
foulds4	-8	-4	50.0%	0.00%
foulds5	-8	-3	62.5%	0.00%
bental4	-450	-450	0.0%	0.00%
bental5	-3500	-900	74.3%	0.00%
haverly1	-400	-400	0.0%	0.00%
haverly2	-600	-400	33.3%	0%
haverly3	-750	-750	0.0%	0%

Table 6.17: Applying CONOPT to pooling problems

6.9 Comparing Grid-based and Pattern-based Models

We now compare the pattern-based and the grid-based models for selected MINLP problems. We use four-segment rectangular grids for this comparison and solve the resulting MIPs to optimality.

Problem	Adjacency	Grid-based			Pattern-based		
		MIP size (cont/bin/ constr)	MIP nodes	MIP time	MIP size (cont/bin/ constr)	MIP nodes	MIP time
ex1252	Seg.	2560/219/896	13653	3.8	2455/135/644	5799	1.2
	Cum.	2560/219/794	6034	1.1	2455/135/584	3400	0.6
	SOS-2	2560/15/386	175693	22.4	2455/15/344	18064	3.6
	Log	2560/117/590	17090	5.0	2455/75/464	6776	1.2
ex1252a	Seg.	2557/207/887	14242	3.5	2452/123/635	5408	1.0
	Cum.	2557/207/785	7211	1.2	2452/123/575	3338	0.7
	SOS-2	2557/3/377	1262429	101.0	2452/3/335	51259	7.2
	Log	2557/105/581	20101	4.5	2452/63/455	8664	1.4
ex1263	Seg.	597/200/616	1697	0.6	537/152/472	1780	0.6
	Cum.	597/200/552	387	0.4	537/152/432	422	0.4
	SOS-2	597/72/296	2072	0.4	537/72/272	3763	0.4
	Log	597/136/424	983	0.4	537/112/352	1821	0.5
ex1264	Seg.	597/196/616	1003	0.5	537/148/472	769	0.4
	Cum.	597/196/552	625	0.4	537/148/432	640	0.3
	SOS-2	597/68/296	5240	0.6	537/68/272	3653	0.4
	Log	597/132/424	3744	0.8	537/108/352	734	0.4

Table 6.18: Comparing grid-based and pattern-based models

Problem	Adjacency	Grid-based			Pattern-based		
		MIP size (cont/bin/ constr)	MIP nodes	MIP time	MIP size (cont/bin/ constr)	MIP nodes	MIP time
ex1265	Seg.	931/300/950	2361	0.9	831/220/710	815	0.5
	Cum.	931/300/850	295	0.5	831/220/650	0	0.3
	SOS-2	931/100/450	7910	1.0	831/100/410	5494	0.9
	Log	931/200/650	1725	0.6	831/160/530	838	0.4
synheat	Seg.	1320/140/624	338	0.5	1300/124/576	235	0.6
	Cum.	1320/140/560	0	0.2	1300/124/520	0	0.3
	SOS-2	1320/12/304	475	0.5	1300/12/296	0	0.3
	Log	1320/76/432	0	0.3	1300/68/408	267	0.5

Table 6.18 contd.: Comparing grid-based and pattern-based models

1) As expected, a pattern-based model is smaller and easier to solve than the corresponding grid-based model.
2) We found that all the four types of grid-based models have the same LP relaxation value. Similary, all the four types of pattern-based models have the same LP relaxation value. In fact, we prove in Appendix A11 that the MIP model with segment-wise adjacency is LP-equivalent to the MIP model with cumulative adjacency. Further, for each of the problems in Table 6.18, the LP value at the root of the branch and bound tree (before CPLEX does any preprocessing or adds any cuts), the LP value of the grid-based model is equal to the LP value of the pattern-based model. However, we found that in many cases, CPLEX is able to generate better cuts for the pattern-based models than for the grid-based models and therefore the LP

value after pre-processing and cuts is higher for a pattern-based model than for the corresponding grid-based model. Finally, in most cases, a pattern-based model requires fewer branch and bound nodes and less time than required by the corresponding grid-based model.

3) In most cases, the pattern-based model with cumulative adjacency does the best both in terms of number of MIP nodes and the MIP time.

6.10 Comparing Pattern-based and Combined Partition Models

We now compare the performance of the pattern-based model with that of a combined partition model for a specific reformulation of the gasnet problem. In this reformulation, we use a 2-segment grid for four-dimensional functions and 4-segment grids for two-dimensional functions. Also, we removed/updated vertex variables using constraint and function information for these runs. The time limit was set at 600 seconds and we enabled variable elimination using external bounds.

		Pattern-based Model			**Combined Partition Model**		
Problem	**Adjacency**	**LP value at root**	**MIP Time**	**MIP Gap**	**LP value at root**	**MIP Time**	**MIP Gap**
gasnet	Seg.	834983.6	600	1.12%	834984.0	483	0.00%
	Cum.	834983.6	358	0.00%	834984.0	147	0.00%
	SOS-2	834983.6	600	56.67%	834984.0	600	31.62%
	Log	823097.9	600	14.67%	823097.9	600	14.69%

Table 6.19: Comparing pattern-based and combined partition models

1) We find that the log model has lower LP value at the root node. This is due to the variable reduction strategies that we have enabled in these runs. If we do not emply these strategies, then the LP value at the root node is the same for all the four models.
2) A combined partition model has a slightly higher LP relaxation value than that of the corresponding combined partition model.
3) The combined partition model does better than the corresponding pattern-based model in terms of the overall MIP solution time and the MIP gap at termination.

6.11 Effect of Problem Reduction and Problem Strengthening Techniques

The problem reduction strategies use external bounds on the independent variables of nonlinear function to either eliminate vertex-weight variables in the PLA-based MIP or set a bound on certain vertex based variables that are adjacenct to infeasible vertices. Problem strengthening strategies involve adding the constraint-based inequalities to the model. Table 6.20 summarizes the effects of these enhacements on three different reformulations of *gasnet*, which is the hardest problem in our problem set. Reformulations *gasnet*-1, *gasnet*-2, and *gasnet*-3 (which correspond to Model-1, Model-2 and Model-3 in Appendix A9) differ in terms of the types of constraints present in the model. Speciafically, *gasnet*-1, *gasnet*-2, and *gasnet*-3 contain constraints of the form $x - y \leq 0$, $x + y \leq 0$, and $x + y \geq 0$ respectively. These reformulations help us demonstrate the effectiveness of different types of valid inequalities for each of these constraint types.

Problem	**Problem Reduction**	**Constraint -based inequalitie s**	**Vertex vars modified**	**Vertex vars deleted**	**LP value at root**	**LP value after cuts**	**MIP Time (CPU s)**	**MIP Gap**	**Rel. Local Global Obj. Gap η**
gasnet-1	No	No	0	0	811309	2081203	1806	11.6%	0.66%
	Yes	No	2393	2614	2128123	2214295	1806	5.4%	0.66%
	No	Yes	0	0	811309	2242709	882	0.0%	0.07%
	Yes	Yes	2393	2614	2130191	2312883	1042	0.0%	0.66%
gasnet-2	No	No	0	0	811309	2098723	1805	15.5%	1.77%
	Yes	No	2393	2614	2128123	2293765	1807	3.9%	0.07%
	No	Yes	0	0	811309	2117416	1018	0.0%	0.66%
	Yes	Yes	2393	2614	2159649	2319815	1234	0.0%	0.07%
gasnet-3	No	No	0	0	811309	1907000	1808	13.9%	0.07%
	Yes	No	2393	2614	2128123	2253946	1806	5.2%	0.07%
	No	Yes	0	0	811309	2241496	969	0.0%	0.07%
	Yes	Yes	2393	2614	2128185	2321628	1098	0.0%	0.00%

Table 6.20: Effect of problem strengthening strategies

1) We find that the problem reduction strategy increases the LP relaxation value at the root node, which indicates that the original LP solution was assigning a positive weight to a number of infeasible vertices.
2) The constraint-based inequalities do not improve the LP relaxation value of the mixed integer program. However, they help Cplex generate better cuts for the problems, thereby leading to a higher value of the LP value after preprocessing and cuts, depicted in the seventh column in Table 6.20.

3) Using both the problem reduction strategies and the constraint-based inequalities is more effective than using only one of these two techniques in terms of the LP relaxation value after preprocessing and cuts.
4) For this particular problem, the constraint-based inequalities seem to do better in the absence of problem reduction strategies than when used in conjunction with problem reduction.

6.12 Comparison with DICOPT

We now compare the performance of the PLA approach with that of DICOPT using the default DICOPT settings and with a specific starting point of zero. We use the MINLP-Lib problems for this comparison.

Problem	Global Obj. Value	DICOPT		
		Objective Value	Absolute gap	Relative gap
ex1221	7.67	NA	NA	13%
ex1243	83402.51	421713.71	338311.21	406%
ex1244	82042.91	87646.38	5603.48	7%
ex1252	128893.74	NA	NA	NA
ex1252a	128893.74	NA	NA	NA
ex1263	19.60	20.60	1.00	5%
ex1263a	19.60	21.00	1.40	7%
ex1264	8.60	9.30	0.70	8%
ex1264a	8.60	10.30	1.70	20%
ex1265	10.30	15.10	4.80	47%
ex1265a	10.30	NA	NA	NA
ex1266	16.30	NA	NA	NA
ex1266a	16.30	NA	NA	NA
gasnet	6999381.56	7045326.89	45945.33	1%

Table 6.21: MINLP-Lib problems that could not be solved by DICOPT

For 6 of the 14 MINLP problems, DICOPT could not find a feasible solution. For the other eight problems, DICOPT could find a feasible solution but not the global optimal solution. Except for *ex1263a* and *ex1264a*, the PLA method could solve all the other MINLP problems. Further, as shown in Table 6.12, the PLA approach could solve *ex1263a* to gap of 2% and *ex1264a* to a gap of 3%. Thus, the PLA approach can do much better than DICOPT.

Chapter 7: Conclusions

Piecewise linear approximation is a promising global optimization technique that works well on a broad range of nonlinear nonconvex problems. Although this approach does not provide a guarantee of optimality, yet, as we demonstrate, it does provide good solutions for a large number of problems. Solvers which work in a branch and bound framework need a lot more time to solve a problem since they have to solve a subproblem at every node of the branch and bound tree. Thus, if one just needs a good-enough solution quickly, then one might consider using the PLA approach. The PLA-based method is particularly suited to mixed integer nonlinear optimization problems which are among the hardest nonlinear non-convex optimization problems.

In our study, we have demonstrated the effectiveness of the PLA apprpoach and proposed various enhancements that can make this approach a powerful global optimization strategy. There are a number of avenues for future research. First, as of now, the non-uniform grid generation method can only deal with two dimensional grids. It might be interesting to see whether this method can scale up to three or higher-dimensional functions. Second, in our study, we have only examined the Union Jack and Crisscross triangulations. However, there might be functions where other triangulations such as H and K could provide better approximation qualities than that provided by the Union Jack triangulation. In fact, the Union Jack triangulation, which can be seen as a hybrid of the H and K triangulations, will have an inferior approximation quality than either the H or the K triangulation for many functions including the bilinear function. The question is how to develop a mixed integer programming model for these triangulations that is as parsimonious as the Union Jack triangulation. Third, the current set of constraint-based valid inequalities can only handle two-variable constraints. It might be interesting to develop constraint-based valid inequalities using constraints with more than two variables. Also, the current constraint-based inequalities are applicable to a non-logarithmic PLA model. It should be possible to extend them to logarithmic PLA models. Fourth, there are problems that involve nonlinear functions of pure integers. It seems that specialized models could be developed for applying the PLA approach to such functions. Lastly, a more ambitious goal will be to compute, within the PLA framework, the convex envelopes for the various functions and thereby have a PLA-based method that not only gives a solution but also provides an optimality gap for that solution. This would help us quantify the quality of the PLA-based solutions and then benchmark the PLA-approach against solvers such as BARON (Ryoo and Sahinidis 1996).

Appendix A1: Proof of Proposition 1

<u>Proposition1</u>: For the bilinear function $f(x, y) = xy$, a Crisscross triangulation with uniform segments along both axes provides the same approximation quality as that provided by a Union Jack triangulation with twice the number of uniform segments along both axes.

<u>Proof</u>: Consider the bilinear function $f(x, y) = xy$, $x \in [a_1, a_3]$, $y \in [b_1, b_3]$. Let us approximate this function by a Crisscross triangulation with break points $\{a_1, a_3\}$ and $\{b_1, b_3\}$ (along the two axes) and by a Union Jack triangulation with equally spaced break points $\{a_1, a_2, a_3\}$ and $\{b_1, b_2, b_3\}$ along the two axes. (Refer Figure A1)

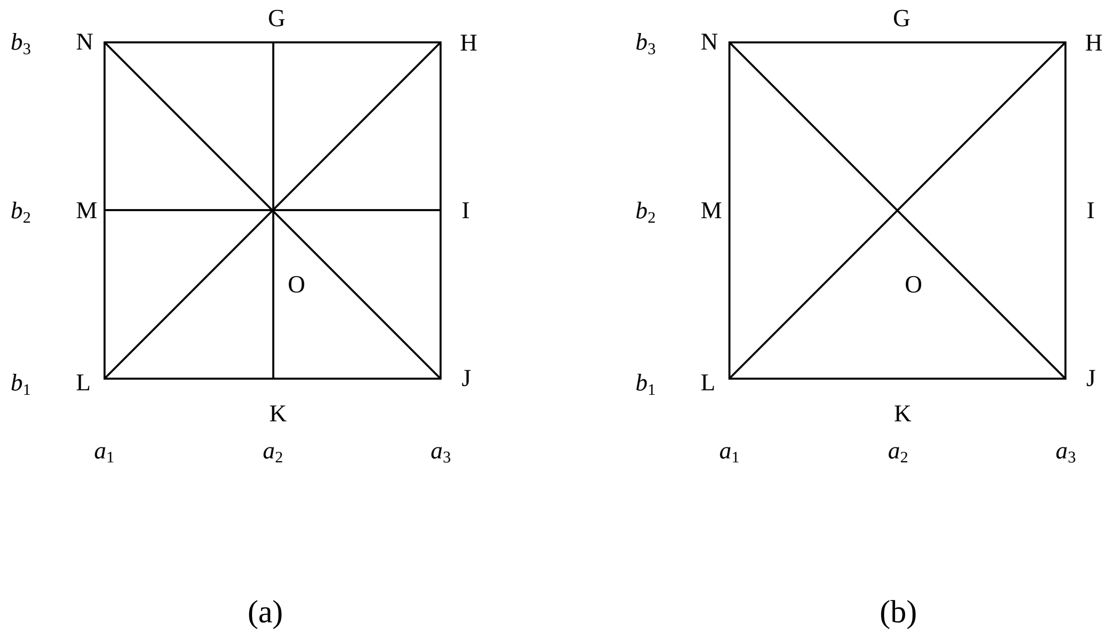


Figure A1.1: Union Jack triangulation versus Crisscross triangulation

Let $L(a_1, b_1)$, $M(a_1, b_2)$, $N(a_1, b_3)$, and $O(a_2, b_2)$ be grid points lying in the domain of f. Since the segments are equally spaced, M is equidistant from L and N. Let $L'(a_1, b_1, f(a_1, b_1))$, $M'(a_1, b_2, f(a_1, b_2))$, $N'(a_1, b_3, f(a_1, b_3))$, and $O'(a_2, b_2, f(a_2, b_2))$ be points in three dimensional space that represent the function values at points L, M, N and O respectively. Consider the three dimensional plane formed by L', O' and N'. Let the equation of this plane be $px + qy + rz + s = 0$. Since L' and N' lie on this plane, therefore

$$pa_1 + qb_1 + r\, f(a_1, b_1) + s = 0 \qquad \text{(A1.1)}$$

$$pa_1 + qb_3 + r\, f(a_1, b_3) + s = 0. \qquad \text{(A1.2)}$$

Adding Equation (A1.1) and Equation (A1.2) and dividing by 2, we get:

$$pa_1 + q(b_1 + b_3)/2 + r[\, f(a_1, b_1) + f(a_1, b_3)\,]\,/2 + s = 0. \qquad \text{(A1.3)}$$

Since M is equidistant from L and N, $b_2 = (b_1 + b_3)/2$. Therefore we get

$$pa_1 + qb_2 + r[\, f(a_1, b_1) + f(a_1, b_3)\,]\,/2 + s = 0. \qquad \text{(A1.4)}$$

$M'(a_1, b_2, f(a_1, b_2))$ will lie on the plane through L', O' and N' if

$$pa_1 + qb_2 + r\, f(a_1, b_2) + s = 0 \qquad \text{(A1.5)}$$

or

$$f(a_1, b_2) = [\, f(a_1, b_1) + f(a_1, b_3)\,]\,/2. \qquad \text{(A1.6)}$$

Since the bilinear function satisfies the condition given by Equation (A1.6), M' lies on the plane through L', O' and N'. Therefore, the quality of approximation provided by ΔLOM and ΔMON is the same as the quality of approximation provided by ΔLON. The same argument holds for ΔLOJ, ΔJOH and ΔHON.

Appendix A2: Algorithm for One-dimensional Shortest Path-based Method

This algorithm takes as input a one-dimensional function, a set of break points that divide the function domain into intervals and the number of segments into which we wish to partition the domain. It returns a set of break points that partition the domain into no more than the desired number of segments and have the lowest approximation error as computed by pre-specified metrics for the arc and the path costs. The algorithm is composed of two procedures: *create graph*, which specifies how to create the directed graph, and *get break points*, which describes how to obtain a set of break points that correspond to the least cost path in the graph such that the path contains no more than a fixed number of arcs.

Notation

f	nonlinear function for which a piecewise linear approximation is required
n	no. of equal-sized intervals into which we initially divide the domain
H	number of segments required
X	set of break points given as an input
Ω	set of sample points given as an input
$G = (N, A)$	directed graph with node set N and arc set A
i, j, k	index over N
a	index over A
ω	index over Ω
c_{ij}	cost of arc (i, j)
x_i, x_j	break points corresponding to nodes i and j
$\Omega(i, j)$	sample points lying between break points x_i and x_j

$f(i)$	true function value at break point x_i
$f(\omega)$	true function value at sample point ω
$\phi(\omega)$	approximate function value at sample point ω
$Conv(\omega, f, i, j)$	convex combination of the function values at break points x_i and x_j using weights that express ω as a convex combination of x_i and x_j
ε_A	metric for computing arc costs, is either *Sum of Errors* or *Maximum Error*
ε_P	metric for computing path cost, is either *Sum of Arc Costs* or *Maximum Arc Cost*
$L(j, h)$	Length of the shortest path from node 0 to node *j* containing no more than *h* arcs (the shortest *h*-hop path)
$Pred(j, h)$	Predecessor node of node *j* in the shortest *h*-hop path to node *j*
B^*	set of break points that provide the best approximation
π	cost of the shortest path

Algorithm A1: *get best segments* $(n, H, \varepsilon_A, \varepsilon_P, f)$

```
begin
    $G$ = create graph $(n, \varepsilon_A, f)$
    $[B^*, \pi]$ = get break points $(G, n, H, \varepsilon_P)$
end
```

Procedure P1: *create graph* (n, ε_A, f)

```
begin
    Create node set $N = \{0, 1, \ldots, n\}$
    Add arcs to arc-set A, i.e., add arc($i, j$) such that
        $i = 0, 1, \ldots, n - 1$
        $j = i + 1, i + 2, \ldots, n$
    for all $(i, j) \in A$ do
        $\sigma := 0$
        $\mu := -\infty$
        for $\omega \in \Omega(i, j)$ do
                $\phi(\omega) := Conv(\omega, i, j, f)$
                $\delta := |f(\omega) - \phi(\omega)|$
                $\sigma := \sigma + \delta$
                if $\mu < \delta$ then $\mu := \delta$
                end if
        next $\omega$
        if $\varepsilon_A$ = Sum of Errors then $c_{ij} := \sigma$
        else if $\varepsilon_A$ = Maximum Error then $c_{ij} := \mu$
    end if
  next $(i, j)$
  $G := (N, A)$
return G
end procedure
```

Procedure P2: *get break points* (G, n, H, ε_P)

```
begin
    $L(j, 1) := c_{0j}$        for $j = 1, \ldots, n$
    $Pred(j, 1) := 0$        for $j = 1, \ldots, n$                                    (Initialization)
    for $h = 2, \ldots, H$ do
        for $j = 1, \ldots, n$ do
            $L(j, h) := L(j, h-1)$
            $Pred(j, h) := Pred(j, h-1)$
            if $j \geq h$ then                                                (Loop 1)
                for $i = h-1$ to $j-1$ do
                    if $\varepsilon_P$ = Sum of Arc Costs
                        if $L(i, h-1) + c_{ij} < L(j, h)$ then
                            $L(j, h) := L(i, h-1) + c_{ij}$
                            $Pred(j, h) := i$
                        end if
                    else if $\varepsilon_P$ = Maximum Arc Costs
                        if $\mathrm{Max}(L(i, h-1), c_{ij}) < L(j, h)$ then
                            $L(j, h) := \mathrm{Max}(L(i, h-1), c_{ij})$
                            $Pred(j, h) := i$
                        end if
                    end if
                next $i$
            end if
        next $j$
    next $h$
    $j := n$
    $h := m$
    while $j \neq 0$ do
        add $x_j$ to $B^*$
        $j := Pred(j, h)$
        $h := h - 1$
    end while
end procedure
```

---

1) At the end of stage *h*, the algorithm obtains the shortest paths of length at most *h* from the source to all the nodes.
2) In the initialization phase, we identify the one-hop path to every node, which is simply the arc connecting the source node to that node.
3) Since the path from the source node to node *j* can have at most *j* arcs, $L(j, h+1)$ will be equal to $L(j, h)$ for all *h* greater than or equal to *j* . Thus, the code in Loop 1 is executed only if *j* is greater than *h*. This observation makes the code more efficient, though it does not improve the worst-case complexity of the procedure.

4) We recover the nodes on the shortest path by starting with the sink node and iteratively identifying the predecessors until we reach node 0.
5) The computational complexity of the algorithm is $O(n^2H)$ where the number of arcs in the graph is $O(n^2)$.

Appendix A3: Alternating Shortest Path-based Method for Crisscross grids

This algorithm takes as input a two-dimensional function, a set of break points along the horizontal and vertical axis and the desired number of segments along each axis. It returns a set of break points along the two axes such that along each axis, the number of segments is less than the desired number of segments along that axis and the segments yield a Crisscross triangulation that has the lowest approximation error as computed by pre-specified metrics for the arc and the path costs. The algorithm, which we name *get best non-uniform grid* calls three procedures: *create graph X-axis, create graph Y-axis* and *get break points*. (See Figure A2 for a schematic)

1. The *create graph X-axis* procedure takes as inputs the break points along the Y-axis and creates a directed graph over which a shortest path problem could be solved to get a set of good break points along the X-axis. This procedure internally calls the procedure *get error vertical pattern* to compute the arc costs in the directed graph.
2. The *create graph Y-axis* procedure takes as inputs the break points along the X-axis and creates a directed graph over which a shortest path problem could be solved to get a set of good break points along the Y-axis. This procedure internally calls the procedure *get error horizontal pattern* to compute the arc costs in the directed graph.
3. The procedure *get* b*reak points* is the procedure that we used to solve a shortest path problem over a directed network corresponding to a one dimensional function.

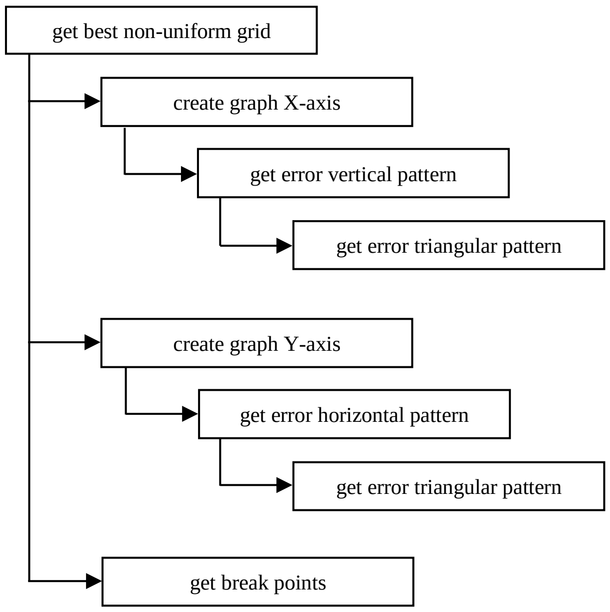

get best non-uniform grid
create graph X-axis
get error vertical pattern
get error triangular pattern
create graph Y-axis
get error horizontal pattern
get error triangular pattern
get break points

Figure A3.1: Procedures for alternating shortest-path-based method

The procedures *get error vertical pattern* and *get error horizontal pattern* internally call the procedure *get error triangular pattern* which gives the approximation error for a triangular pattern corresponding to a combination of a horizontal break point pair and a vertical break point pair.

<u>Notation</u>

f	nonlinear function that has to be approximated
n_X	no. of intervals into which we initially divide the domain along the X-axis
n_Y	no. of into which we initially divide the domain along the Y-axis
h_X	no. of segments required along the X-axis
h_Y	no. of segments required along the Y-axis

X	set of break points along the X-axis	=	$\{x_0, x_1, x_2, ..., x_{n_X}\}$
Y	set of break points along the Y-axis	=	$\{y_0, y_1, y_2, ..., y_{n_Y}\}$
B_X	subset of break points along the X-axis		
B_Y	subset of break points along the Y-axis		
B_X^*	best subset of break points along the X-axis		
B_Y^*	best subset of break points along the Y-axis		
Ω	set of sample points		
P	set of grid points		
ω	index over Ω		
i, j	indices over X		
k, l	indices over Y		
$p(i,k)$	grid point corresponding to break points x_i and y_k along the X and Y axes		
p_{ijkl}^{mid}	grid point corresponding to intersection of the diagonals of the rectangle formed by grid points $p(i,k)$, $p(j,k)$, $p(j,l)$, $p(i,l)$.		
$\Omega(r,s)$	sample points lying in the rectangle defined by break point pair r along the X-axis and break point pair s along the Y-axis		
$f(p)$	true function value at grid point p		
$f(\omega)$	true function value at sample point ω		
$\phi(\omega)$	approximate function value at sample point ω		
p_1, p_2, p_3	Indices over P		

$Conv(\omega, p_1, p_2, p_3, f)$	convex combination of the function values at grid points p_1, p_2 and p_3 using weights that express ω as a convex combination of p_1, p_2 and p_3
ε_A	metric for computing arc costs, is either *Sum of Errors* or *Maximum Error*
ε_P	metric for computing path cost, is either *Sum of Arcs Costs* or *Maximum Arc Cost*
$G = (N, A)$	directed graph with node set N and arc set A
a	index over A
c_{ij}	cost of arc (i, j)
$L(j, h)$	Length of the shortest path from node 0 to node j containing no more than h arcs (the shortest h-hop path)
$Pred(j, h)$	Predecessor node of node j in the shortest h-hop path to node j
π	cost of the shortest path
π^*	current best approximation cost for the algorithm

Algorithm A2: *get best non-uniform grid* $\left(\text{initial axis}, n_X, n_Y, h_X, h_Y, \varepsilon_A, \varepsilon_P, f\right)$

```
begin
    if initial axis = X-axis then
        Populate $B_Y^{cur}$ with uniform break points along the Y-axis
        current axis : = X-axis
    else if initial axis = Y-axis then
        Populate $B_X^{cur}$ with uniform break points along the X-axis
        current axis : = Y-axis
    end if
    $\pi^* := \infty$
    continue flag : = TRUE
    while continue flag = TRUE do
        if current axis = X-axis then
            $G :=$ create graph X-axis $\left(n_X, B_Y, P, \Omega, \varepsilon_A, f\right)$
            $\left[B_X, \pi\right] :=$ get break points $\left(G, n_X, h_X, \varepsilon_P\right)$
            if $\pi < \pi^*$ then
                $B_X^* := B_X$
                $\pi^* := \pi$
                current axis : = Y-axis
            else continue flag : = FALSE
            end if
        else if Current axis = Y-axis then
            G : = create graph Y-axis $\left(n_Y, B_X, P, \Omega, \varepsilon_A, f\right)$
            $[B_Y, \pi]$: = *get best break points* $(G, n_Y, h_Y, \varepsilon_P)$
            **if** $\pi < \pi^*$ **then**
                $B_Y^* := B_Y$
                $\pi^* := \pi$
                current axis : = Y-axis
            **else** continue flag : = FALSE
            **end if**
        **end if**
    **end while**
    **return** $(B_X^*, B_Y^*)$
**end procedure**
```

Procedure P3: *create graph X-axis* $(n_X, B_Y, P, \Omega, \varepsilon_A, f)$

```
begin
        Create node set $N = \{0, 1, ..., n_X\}$ corresponding to break points $\{x_0, x_1, x_2, ..., x_{n_X}\}$
        Add arc($i$, $j$) to arc set $A$ such that
            $i = 0, 1, \ldots, n_X - 1$
            $j = i + 1, i + 2, \ldots, n_X$
            $c_{ij} :=$ *get error vertical pattern* $(i, j, B_Y, P, \Omega, \varepsilon_A, f)$
        $G$: = ($N$, $A$)
        return $G$
```

```
end procedure
```

```
Procedure P4: create graph Y-axis $(n_Y, B_X, P, \Omega, \varepsilon_A, f)$
begin
    Create node set $I = \{0,1,...,n_Y\}$ corresponding to break points $\{y_0, y_1, y_2,..., y_{n_Y}\}$
    Add arc(k, l) to arc set A such that
        $k = 0, 1, \ldots, n_Y - 1$
        $l = i + 1, i + 2, \ldots, n_Y$
        $c_{kl}$ := get error horizontal pattern$(k, l, B_X, P, \Omega, \varepsilon_A, f)$
    $G$: = $(N, A)$
    return $G$
end procedure
```

Procedure P6: *get error vertical pattern* $(i, j, B_Y, P, \Omega, \pi_A, f)$

```
begin
    $\sigma := 0$
    $\mu := -\infty$
    for $b = 1 \text{ to } |\mathrm{B_Y}| - 1$ do
        $\delta :=$ get error triangular pattern$(i, j, B_Y[b], B_Y[b+1], P, \Omega, \varepsilon_A, f)$
        $\sigma := \sigma + \delta$
        if $\mu < \delta$ then $\mu := \delta$
        end if
    next b
    if $\varepsilon_A$ = Sum of Errors then pattern error : = $\sigma$
    else if $\varepsilon_A$ = Maximum Error then pattern error : = $\mu$
    end if
end
return pattern error
end procedure
```

Procedure P7: *get error horizontal pattern* $(k, l, B_X, P, \Omega, \varepsilon_A, f)$

```
begin
    $\sigma := 0$
    $\mu := -\infty$
    for $b = 1 \text{ to } |\mathrm{B_X}| - 1$ do
```

```
        $\delta :=$ get error triangular pattern$\left(B_X[b], B_X[b+1], k, l, P, \Omega, \varepsilon_A, f\right)$
        $\sigma := \sigma + \delta$
        **if** $\mu < \delta$ **then** $\mu := \delta$
        **end if**
    **next b**
    **if** $\varepsilon_A$ = Sum of Errors **then** pattern error : = $\sigma$
    **else if** $\varepsilon_A$ = Maximum Error **then** pattern error : = $\mu$
    **end if**
    **return** pattern error
**end procedure**
```

Procedure P8: *get error triangular pattern* $\left(i, j, k, l, P, \Omega, \varepsilon_A, f\right)$

```
begin
    Define grid points $p_1, p_2, p_3, p_4,$ and $p_5$ such that
            $p_1 = p(i,k)$
            $p_2 = p(j,k)$
            $p_3 = p(j,l)$
            $p_4 = p(i,l)$
            $p_5 = p_{ijkl}^{mid}$
    $\sigma := 0$
    $\mu := -\infty$
    for all $\omega \in \Omega(r,s)$ do
        if $\omega$ lies in $\Delta(p_1 p_2 p_5)$ then $\phi(\omega) := Conv(\omega, p_1, p_2, p_5, f)$
        else if $\omega$ lies in $\Delta(p_2 p_3 p_5)$ then $\phi(\omega) := Conv(\omega, p_2, p_3, p_5, f)$
        else if $\omega$ lies in $\Delta(p_3 p_4 p_5)$ then $\phi(\omega) := Conv(\omega, p_3, p_4, p_5, f)$
        else if $\omega$ lies in $\Delta(p_4 p_1 p_5)$ then $\phi(\omega) := Conv(\omega, p_4, p_1, p_5, f)$
        end if
        $\delta := |f(\omega) - \phi(\omega)|$
        $\sigma := \sigma + \delta$
        if $\mu < \delta$ then $\mu := \delta$
        end if
        next $\omega$
    end if
```

```
        if $\varepsilon_A$ = Sum of Errors then pattern error : = $\sigma$
        else if $\varepsilon_A$ = Maximum Error then pattern error : = $\mu$
        end if
        return pattern error
    end procedure
```

1) In procedure *get error triangular pattern*, we consider all the sample points that lie within the rectangle formed by the vertical and the horizontal break point pairs. For each point, we compute the approximate function value using the triangulation shown in Figure A3. We then compute the approximation error for each sample point, which is the absolute value of the difference between the true and the approximate function value for the sample point.

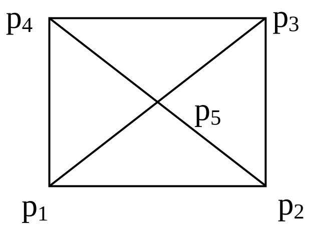


Figure A3.2: Computing approximation error for a triangular pattern

2) Depending upon the whether the error metric is *Sum of errors* or *Maximum Error*, the approximation error for the entire pattern is either the sum or the maximum of the approximation errors for all the sample points lying within that pattern.

Appendix A4: Alternating Shortest Path-based Method for Union Jack Grids

To apply the alternating shortest path method for obtaining Union Jack grids, we have to view this grid in terms of its building blocks. We first observe that the Union Jack triangulation can have two different orientations, which we call *UJ-I* and *UJ-II*. (Figure A4.1)

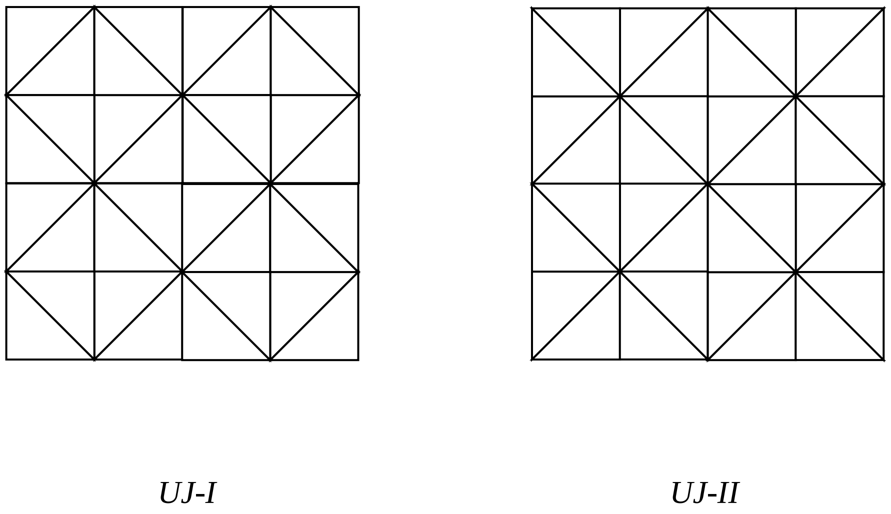


Figure A4.1: Orientations of the two dimensional Union Jack triangulation

The smallest building blocks of this grid are the two basic types of *triangular patterns*, which we call *UJ-T1* and *UJ-T2*. We show these patterns in Figure A4.2.

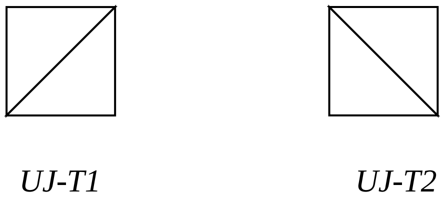


Figure A4.2: Triangular patterns for a two dimensional Union Jack triangulation

A horizontal sequence of alternating types of triangular patterns gives rise to what we call a horizontal pattern. Depending upon the type of the starting triangular pattern, we can have two kinds of horizontal patterns, which we call *UJ-H1* and *UJ-H2*. Similarly, a vertical sequence of alternating types of triangular patterns gives rise to a vertical pattern, which also can have two types, *UJ-V1* and *UJ-V2*, depending upon the type of the starting triangular pattern (Figure A4.3). A Union jack grid can be visualized either as a sequence of alternating vertical patterns or as a sequence of alternating horizontal patterns. Since each pattern is followed by a pattern of a different type, the starting pattern determines the orientation of the entire grid. Thus, a *UJ-I* grid starts with a *UJ-H1* horizontal pattern at its base or a *UJ-V1* vertical pattern at its extreme left. On the other hand, a *UJ-II* grid starts with a *UJ-H2* horizontal pattern or a *UJ-V2* vertical pattern.

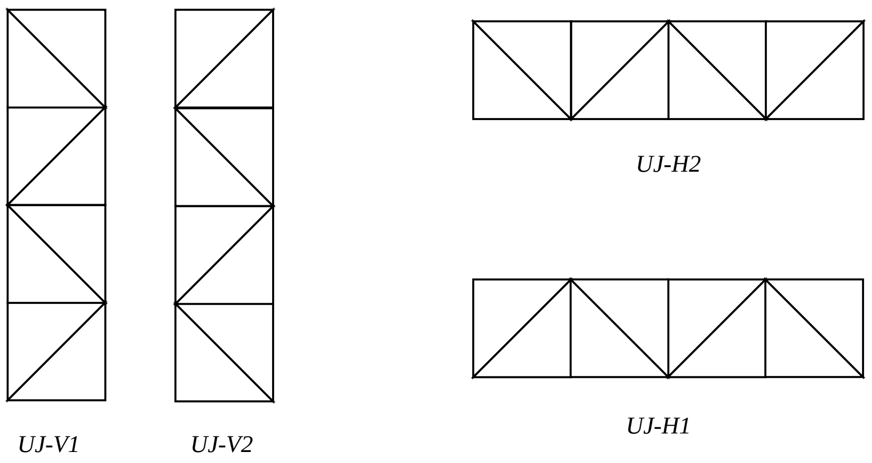
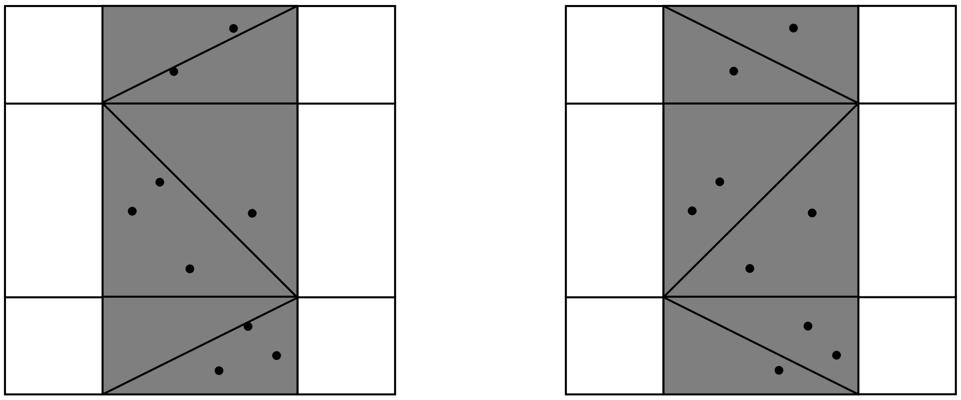


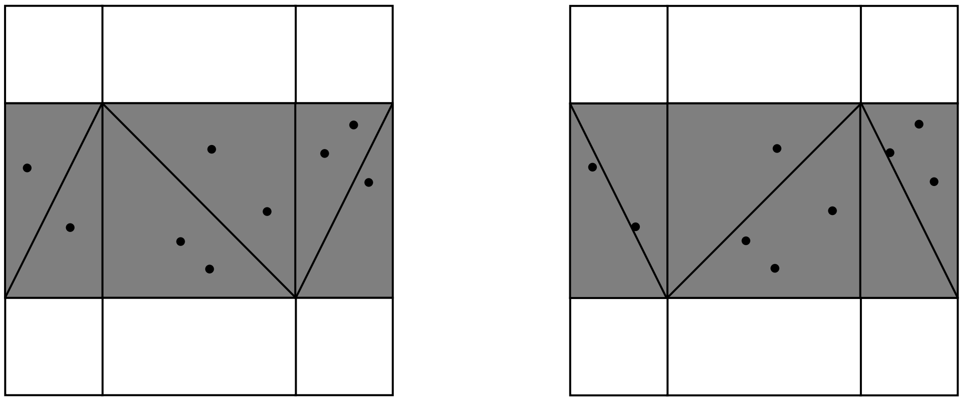
Figure A4.3: Vertical and horizontal patterns for a Union Jack triangulation

To apply the alternating shortest path method to obtain a Union Jack grid, we fix the segments (break points) along one of the axes and then solve a shortest path problem along the other axis to get the best set of (segments) break points along the other axis. However, since each break point pair can have two costs, one for each Union Jack pattern, the network for the shortest path problem in this case is slightly different from the one we used in the one dimensional scenario. We now describe this network.



(a) *UJ-V1* (b) *UJ-V2*

Figure A4.4: Approximation error for a break point pair along horizontal axis



(a) *UJ-H1* (b) *UJ-H2*

Figure A4.5: Approximation error for break point pair along the vertical axis

The nodes of this specially constructed network correspond to the break points. There is one node for each of the two domain end points. The node corresponding to the lower limit is called the source node and the one corresponding to the upper limit is called the sink node. Each of the other break points is associated with two nodes — one Type-A and a Type B node. A Type-A node indicates the beginning of a *UJ-V1* pattern (for the horizontal axis) or the beginning of the *UJ-H1* pattern (for the vertical axis). On the other hand, a Type-B node indicates the beginning of a *UJ-V2* pattern (for the horizontal axis) or the beginning of the *UJ-H2* pattern (for the vertical axis). Similarly, there are two kinds of arcs in the network — Type-A and Type B arcs. Type-A arcs emanate either from the source node or from a Type-A node and end either in the sink node or in a Type-B node that corresponds to a higher-indexed break point. The cost of a Type-A arc is a function of the approximation error for domain points that lie within the horizontal or vertical pattern, assuming that the pattern is of type *UJ-H1* or *UJ-V1* (Figure A4.3). Type-B arcs emanate either from the source node or from a Type-B node and end either in the sink node or in a Type-A node that corresponds to a higher-indexed break point. The cost of a Type-B arc is a function of the approximation error for domain points that lie within the horizontal or vertical pattern, assuming that the pattern is of type *UJ-H2* or *UJ-V2* (Figure A4.3). This function is called the error metric. As in the one dimensional case, we could either have the sum of error or the maximum error as the error metric.

The structure of this graph (Figure A4.6) ensures that in any path from the source node to the sink node, a Type-B arc will always follow a Type-A arc and a Type-A always follows a Type-B arc. In other words, any path from the source node to the sink node would either be an alternating sequence of vertical patterns or an alternating sequence of horizontal patterns, and would thus correspond to a valid Union Jack approximation of the two dimensional function.

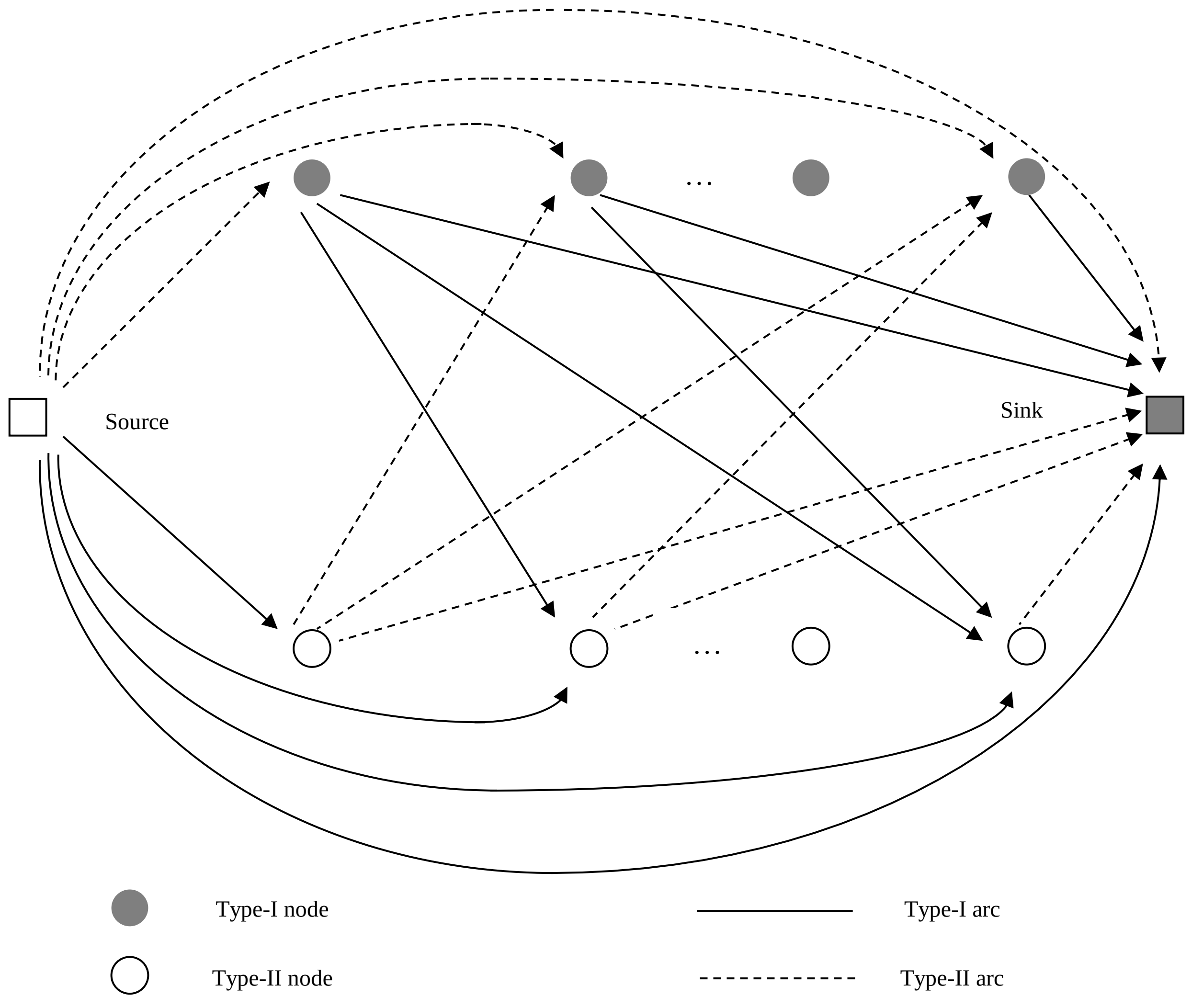


Figure A4.6: Alternating shortest-path-based method for Union Jack grid

Appendix A5: Marginal Weight Consistency Constraints for a Combined Partition

In this appendix, we describe how to obtain the marginal weight consistency constraint for combined partitions. We first recall the notation.

Notation

P	set of patterns for variable *X*
p	index over *P*
n_p	number of segments in partitioning pattern $p \in P$
J_p	set of break points in *p*
m	number of segments in the combined partition for *X*
K	set of break points in the combined partition for *X*; indexed from $k = 0,...,m$ in the order of increasing *X* values
k	index over *K*
σ_j^p	marginal weight variable for break point *j* in pattern $p \in P$, defined for $j = 0,1,...,n_p$
ρ_k	marginal weight variable for breakpoint *k* in the combined partition, defined for $k = 1,...,m$
x_k	value of *X* at break point *k* in the combined partition
x_j^p	value of *X* at break point *j* in partition $p \in P$
$j(k, p)$	closest break point in pattern *p* that either coincides with break point *k* of the combined partition or lies strictly to the right of break point *k* of the combined partition, i.e., $j(k, p) = \text{Min}\, j : x_j^p \geq x_k$, for all $p \in P, k \in K$
$k(j, p)$	break point in the combined partition that coincides with break point *j* in pattern *p*, i.e. , $k(j, p) = k : x_j^p = x_k$, for all $p \in P$, for all $j \in J_p$

$K_p(j)$ set of break points in the combined partition that lie in the interior of segment *j* in partition *p*, defined for all $j \in J_p \setminus \{0\}, p \in P$. In other words, $K_p(j) = \{k(j-1,p)+1, \dots, k(j,p)-1\}$.

We wish to prove the validity of Equation A5.1

$$\sigma_j^p = \rho_{k(j,p)} + \sum_{k \in K_p(j)} \theta_{kp}\, \rho_k + \sum_{k \in K_p(j+1)} (1-\theta_{kp})\, \rho_k \quad \text{for all } p \in P,\ j \in J_p \setminus \{0, n_p\} \qquad \text{(A5.1)}$$

$$\text{where } \theta_{kp} = \frac{x_k - x^p_{j(k,p)-1}}{x^p_{j(k,p)} - x^p_{j(k,p)-1}} \text{ for all } p \in P, k \in K \setminus \{0\} \qquad \text{(A5.2)}$$

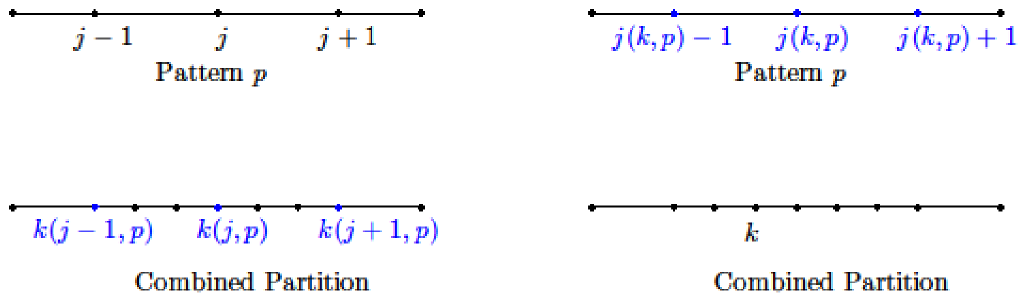


Figure A5.1: Combined partition

Break point *j* in pattern *p* can have a positive weight when the chosen value for *X* either lies in segment *j* or in segment *j*+1.

Let us assume that the integer solution to the PLA model assigns a value *x* to variable *X* and *x* lies in segment *j* of some pattern *p*, i.e. $x_{k(j-1,p)} \le x \le x_{k(j,p)}$. Therefore,

$$x = \rho_{k(j-1,p)} x_{k(j-1,p)} + \sum_{k \in K_p(j)} \rho_k x_k + \rho_{k(j,p)} x_{k(j,p)} \qquad \text{(A5.3)}$$

$$\rho_{k(j-1,p)} + \sum_{k \in K_p(j)} \rho_k + \rho_{k(j,p)} = 1 \qquad \text{(A5.4)}$$

$$\rho_k = 0 \text{ for all } k < k(j-1,p) \text{ and } k > k(j,p) \tag{A5.5}$$

However, any break point k in the combined partition located between $k(j-1,p)$ and $k(j,p)$ can be expressed as a convex combination of $x_{k(j-1,p)}$ and $x_{k(j,p)}$. Therefore,

$$x_k = \theta_{kp} x_{k(j,p)} + (1-\theta_{kp}) x_{k(j-1,p)}. \tag{A5.6}$$

Substituting this value of x_j in Equation (A5.3), we get,

$$x = \rho_{k(j-1,p)} x_{k(j-1,p)} + \sum_{k \in K_p(j)} \rho_k \left(\theta_{kp} x_{k(j,p)} + (1-\theta_{kp}) x_{k(j-1,p)}\right) + \rho_{k(j,p)} x_{k(j,p)} \tag{A5.7}$$

or $$x = \left(\rho_{k(j-1,p)} + \sum_{k \in K_p(j)} \rho_k (1-\theta_{kp})\right) x_{k(j-1,p)} + \left(\rho_{k(j,p)} + \sum_{k \in K_p(j)} \rho_k \theta_{kp}\right) x_{j(j,p)} \tag{A5.8}$$

or $$x = \sigma_{j-1}^p x_{k(j-1)} + \sigma_j^p x_{k(j)} \tag{A5.9}$$

where $\sigma_j^p = \rho_{k(j,p)} + \sum_{j \in K_p(j)} \theta_{kp}\, \rho_k + \sum_{k \in K_p(j+1)} (1-\theta_{kp})\, \rho_k$ for all $p \in P,\ j \in J_p \setminus \{0, n_p\}$.

Appendix A6: Validity of Constraint-based Inequalities

If two variables that appear in two different nonlinear functions are related by a linear constraint, then we can develop cuts that relate the binary segment selection variables and the marginal weight variables across the partitions for *X* and *Y*.

Notation

I:	set of break points in the *X* partition
J:	set of break points in the *Y* partition
i:	index for break points and segments for the *X* partition
j:	index for break points and segments for the *Y* partition
n_X :	number of segments in the *X* partition
n_Y :	number of segments in the *Y* partition
σ_i^X	marginal weight variable for break point *i* along the *X* partition
τ_i^X	cumulative marginal weight variable for break point *i* along the *X* partition
σ_j^Y	marginal weight variable for break point *j* along the *Y* partition
τ_j^Y	cumulative marginal weight variable for break point *j* along the *X* partition
S_i^X	segment selection variable for segment *i* along the *X* partition, which is one if we select segment *i* and is zero otherwise
T_i^X	segment selection variable for segment *i* along the *X* partition, which is one if we select a segment with index *i* or higher, and is zero otherwise
S_j^Y	segment selection variable for segment *j* along the *Y* partition, which is one if we select segment *j* and is zero otherwise
T_j^Y	segment selection variable for segment *j* along the *Y* partition, which is one if we select a segment with index *j* or higher, and is zero otherwise

1. Upper bound on *X* induces an upper bound on *Y*

Consider the situation where *X* and *Y* are related by the constraint $Y \le aX + b$. In this situation, an upper bound on *X* induces an upper bound on *Y*.

Let us assume that we select an *X* segment with index *i* or lower. This implies that *X* cannot exceed x_i. Let $j(i) = \min j : ax_i + b \ge y_j$.Then, we cannot select a *Y* segment whose index is greater than $j(i)$. This gives us the following valid inequalities.

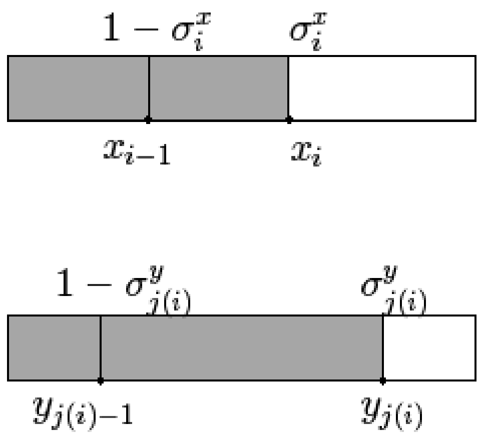


Figure A6.1: Upper bound on *X* inducing upper bound on *Y*

a) Inequalities relating binary variables

Let $\bar{I}_1$ be the set of break points in the *X* pattern such that $I_1 = \{i : 1 \le j(i) \le n_Y - 1\}$. Then, if we select an *X* segment with index *i* or lower, then we must select a *Y* segment with index $j(i)$ or less. So, we have the following inequality.

$$1 - T_{j(i)+1}^{Y} \ge 1 - T_{i+1}^{X} \text{ for every } i \in I_1 \tag{A6.1}$$

b) Inequalities relating marginal weight variables

Let $\bar{y}_i = ax_i + b$. Then, if *X* has an upper bound of x_i, *Y* has an upper bound of $\bar{y}_i$. This implies that there is an upper bound on the marginal weight variable for break point *j*(*i*) along the *Y* partition.

Consider a case where we want to choose

- segment *i* for the *X* variable with weights σ_i^X and $1-\sigma_i^X$ assigned to break points *i* and *i*-1, and
- segment *j*(*i*) for the *Y* variable with weights $\sigma_{j(i)}^Y$ and $1-\sigma_{j(i)}^Y$ assigned to break points *j*(*i*) and *j*(*i*)-1.

Since $Y \le aX + b$, we get the following.

$$\sigma_{j(i)}^Y y_{j(i)} + (1-\sigma_{j(i)}^Y) y_{j(i)-1} \le a(\sigma_i^X x_i + (1-\sigma_i^X) x_{i-1}) + b \qquad \text{(A6.2)}$$

Or $\sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) + y_{j(i)-1} \le a\sigma_i^X x_i + ax_{i-1} - a\sigma_i^X x_{i-1} + b$

Or $\sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) + y_{j(i)-1} \le a\sigma_i^X (x_i - x_{i-1}) + ax_{i-1} + b$

Or $\sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) + y_{j(i)-1} \le a\sigma_i^X (x_i - x_{i-1}) + \bar{y}_{i-1}$ where $\bar{y}_{i-1} = ax_i + b$

Or $\sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) + y_{j(i)-1} \le \sigma_i^X (ax_i - ax_{i-1}) + \bar{y}_{i-1}$

Or $\sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) + y_{j(i)-1} \le \sigma_i^X (\bar{y}_i - \bar{y}_{i-1}) + \bar{y}_{i-1}$ where $\bar{y}_i = ax_i + b$

Or $\sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) \le \sigma_i^X (\bar{y}_i - \bar{y}_{i-1}) + \bar{y}_{i-1} - y_{j(i)-1}$

Or $\sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) \le \sigma_i^X (\bar{y}_i - y_{j(i)-1} - (\bar{y}_{i-1} - y_{j(i)-1})) + \bar{y}_{i-1} - y_{j(i)-1}$

Or $\sigma_{j(i)}^Y \le \sigma_i^X \left(\dfrac{\bar{y}_i - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}} - \dfrac{\bar{y}_{i-1} - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}} \right) + \dfrac{\bar{y}_{i-1} - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}}$ since $y_{j(i)} > y_{j(i)-1}$

Or $\sigma_{j(i)}^Y \le \sigma_i^X (p_i - q_i) + q_i$ where $p_i = \dfrac{\bar{y}_i - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}}$ and $q_i = \dfrac{\bar{y}_{i-1} - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}}$

Or $\tau_{j(i)}^Y \le \tau_i^X (p_i - q_i) + q_i$ since in this case $\sigma_i^X = \tau_i^X$ and $\sigma_{j(i)}^Y = \tau_{j(i)}^Y$ (A6.3)

Thus, if we select segment i for the X variable and assign a cumulative weight of τ_i^X to break point i, then we have an upper bound on the cumulative weight variable for break point $j(i)$. This leads us to the following inequality.

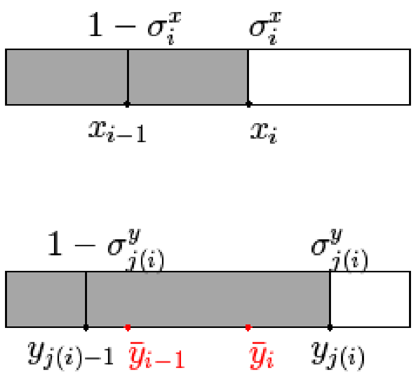
$$\tau_{j(i)}^Y \le \tau_i^X \left(p_i - q_i\right) + q_i(1 - T_{i+1}^X) - q_i(1 - T_{j(i)}^Y) + T_{i+1}^X \qquad \text{(A6.4)}$$

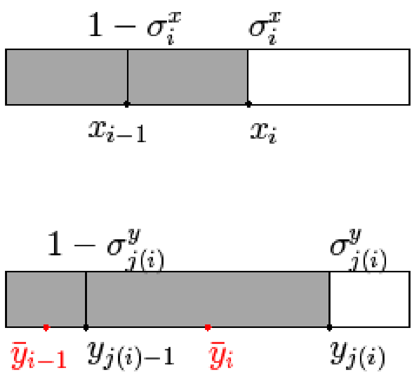
Proof of validity To show that this inequality is valid, we have to consider two cases.

Case 1: $y_{j(i)-1} \le \bar{y}_{i-1}$

Case 2: $y_{j(i)-1} \ge \bar{y}_{i-1}$

We consider four scenarios that cover these two cases. Scenarios 1, 2, 3 and 4 apply to Case 1 and Scenarios 1, 2 and 4 apply to Case 2.





Case 1: $y_{j(i)-1} \le \bar{y}_{i-1}$ Case 2: $y_{j(i)-1} \ge \bar{y}_{i-1}$

Figure A6.2: Scenarios when upper bound on X induces an upper bound on Y

i) Scenario 1: Select X segment with index i and Y segment with index $j(i)$

   This implies $T_{i+1}^X = 0$ and $T_{j(i)}^Y = 1$, therefore $\tau_{j(i)}^Y \le \tau_i^X \left(p_i - q_i\right) + q_i$, which is what we derived in Equation A6.4.

ii) Scenario 2: Select *X* segment with index *i* and *Y* segment with index $j(i)-1$ or lower

This implies $T_{i+1}^{X}=0$ and $T_{j(i)}^{Y}=0$, therefore $\tau_{j(i)}^{Y}\le\tau_{i}^{X}\left(p_i-q_i\right)+q_i(1-0)-q_i(1-0)$ or $\tau_{j(i)}^{Y}\le\tau_{i}^{X}\left(p_i-q_i\right)$. This equation is valid because $\tau_{j(i)}^{Y}$ is equal to zero (because we select a *Y* segment with index $j(i)-1$ or lower) and $\tau_{i}^{X}\left(p_i-q_i\right)\ge 0$ since $p_i\ge q_i$.

iii) Scenario 3: Select *X* segment with index $i-1$ or lower and *Y* segment with index $j(i)$

This implies $T_{i+1}^{X}=0$, $\tau_{i}^{X}=0$ and $T_{j(i)}^{Y}=1$, therefore $\tau_{j(i)}^{Y}\le 0\left(p_i-q_i\right)+q_i(1-0)-q_i(1-1)$ or $\tau_{j(i)}^{Y}\le q_i$ or $\sigma_{j(i)}^{Y}\le q_i$,or $\sigma_{j(i)}^{Y}\le\dfrac{\bar{y}_{i-1}-y_{j(i)-1}}{y_{j(i)}-y_{j(i)-1}}$ which is the bound that one would get on $\tau_{j(i)}^{Y}$ by considering an upper bound of x_{i-1} on the *X* variable. This scenario is valid only for Case 1 above.

iv) Scenario 4: Select *X* segment with index $i-1$ or lower and *Y* segment with index $j(i)-1$ or lower

This implies $T_{i+1}^{X}=0$, $\tau_{i}^{X}=0$ and $T_{j(i)}^{Y}=0$, therefore $\tau_{j(i)}^{Y}\le 0\left(p_i-q_i\right)+q_i(1-0)-q_i(1-0)$ or $\tau_{j(i)}^{Y}\le 0$, which is valid because $\tau_{j(i)}^{Y}$ takes a value of zero since we choose a *Y* segment with index $j(i)-1$ or lower.

2. Upper bound on *X* induces a lower bound on *Y*

Consider the situation where *X* and *Y* are related by the constraint $Y \geq -aX + b$. In this situation, an upper bound on *X* induces a lower bound on *Y*. Let us assume that we select an *X* segment with index *i* or lower. This implies that *X* cannot exceed x_i. Let $j(i) = \max j : -ax_i + b \leq y_j$ .Then, we cannot select a *Y* segment whose index is less than $j(i)+1$. This gives us the following valid inequalities.

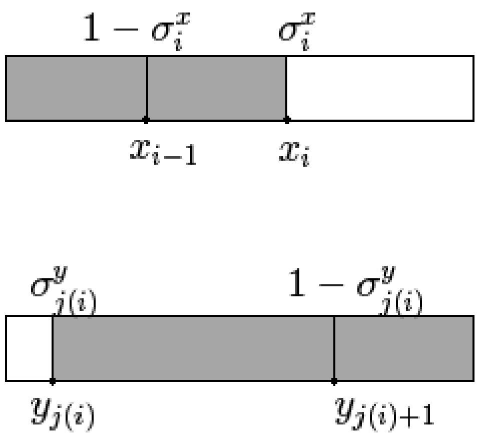


Figure A6.3 Upper bound on *X* inducing a lower bound on *Y*

a) Inequalities relating binary variables

Let $\bar{I}_2$ be the set of break points in the *X* pattern such that $I_2 = \{i : 2 \leq j(i) \leq n_Y - 1\}$. Then, if we select an *X* segment with index *i* or lower, then we must select a *Y* segment with index $j(i)+1$ or higher. So, we have the following inequality.

$$T_{j(i)+1}^{Y} \geq 1 - T_{i+1}^{X} \quad \text{for every } i \in I_2 \tag{A6.5}$$

b) Inequalities relating marginal weight variables

Let $\bar{y}_i = -ax_i + b$. Then, if *X* has an upper bound of x_i, *Y* has a lower bound of $\bar{y}_i$. This implies that there is an upper bound on the marginal weight variable for break point *j*(*i*) along the *Y* partition, and consequently a lower bound on the cumulative marginal weight for break point $j(i)+1$ along the *Y* partition. Consider a case where we want to choose

- segment *i* for the *X* variable with weights σ_i^X and $1-\sigma_i^X$ assigned to break points *i* and *i*-1
- segment $j(i)+1$ for the *Y* variable with weights $\sigma_{j(i)}^Y$ and $1-\sigma_{j(i)}^Y$ assigned to break points $j(i)$ and $j(i)+1$

Since $Y \geq -aX + b$, we get the following.

$$\sigma_{j(i)}^Y y_{j(i)} + (1-\sigma_{j(i)}^Y) y_{j(i)+1} \geq -a(\sigma_i^X x_i + (1-\sigma_i^X) x_{i-1}) + b \qquad \text{(A6.6)}$$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)+1}) + y_{j(i)+1} \geq \sigma_i^X(-ax_i + ax_{i-1}) + (-ax_{i-1} + b)$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)+1}) + y_{j(i)+1} \geq \sigma_i^X(-ax_i + ax_{i-1}) + \bar{y}_{i-1}$ where $\bar{y}_{i-1} = -ax_{i-1} + b$)

Or $\quad \sigma_{j(i)}^Y (y_{j(i)+1} - y_{j(i)}) - y_{j(i)+1} \leq \sigma_i^X(ax_i - ax_{i-1}) - \bar{y}_{i-1}$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)+1} - y_{j(i)}) \leq \sigma_i^X(ax_i - ax_{i-1}) + y_{j(i)+1} - \bar{y}_{i-1}$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)+1} - y_{j(i)}) \leq \sigma_i^X(-ax_{i-1} + b - (-ax_i + b)) + y_{j(i)+1} - \bar{y}_{i-1}$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)+1} - y_{j(i)}) \leq \sigma_i^X(\bar{y}_{i-1} - \bar{y}_i) + y_{j(i)+1} - \bar{y}_{i-1}$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)+1} - y_{j(i)}) \leq \sigma_i^X(y_{j(i)+1} - \bar{y}_i - (y_{j(i)+1} - \bar{y}_{i-1})) + y_{j(i)+1} - \bar{y}_{i-1}$

Or $\quad \sigma_{j(i)}^Y \leq \sigma_i^X \left(\dfrac{y_{j(i)+1} - \bar{y}_i}{y_{j(i)+1} - y_{j(i)}} - \dfrac{y_{j(i)+1} - \bar{y}_{i-1}}{y_{j(i)+1} - y_{j(i)}} \right) + \dfrac{y_{j(i)+1} - \bar{y}_{i-1}}{y_{j(i)+1} - y_{j(i)}}$

Or $\quad \sigma_{j(i)}^Y \leq \sigma_i^X (p_i - q_i) + q_i$ where $p_i = \dfrac{y_{j(i)+1} - \bar{y}_i}{y_{j(i)+1} - y_{j(i)}}$ and $q_i = \dfrac{y_{j(i)+1} - \bar{y}_{i-1}}{y_{j(i)+1} - y_{j(i)}}$

Or $\quad 1 - \tau_{j(i)+1}^Y \leq \tau_i^X (p_i - q_i) + q_i$ since $\sigma_i^X = \tau_i^X$ and $\sigma_{j(i)}^Y = 1 - \tau_{j(i)+1}^Y$

Or $\quad \tau^{Y}_{j(i)+1} \geq (1-q_i) - \tau^{X}_{i}\left(p_i - q_i\right)$ (A6.7)

Thus, if we select segment *i* for the *X* variable and assign a cumulative weight of τ^{X}_{i} to break point *i*, then we have a lower bound on the cumulative weight variable for break point $j(i)+1$. This leads us to the following inequality.

$$\tau^{Y}_{j(i)+1} \geq (1-q_i)(1-T^{X}_{i+1}) - \tau^{X}_{i}\left(p_i - q_i\right) + q_i T^{Y}_{j(i)+2} \qquad \text{(A6.8)}$$

Proof of validity To show that this inequality is valid, we have to consider two cases.

Case 1: $y_{j(i)+1} \geq \bar{y}_{i-1}$

Case 2: $y_{j(i)+1} \leq \bar{y}_{i-1}$

We consider four scenarios that cover these two cases. Scenarios 1, 2, 3 and 4 apply to Case 1 and Scenarios 1, 2 and 4 apply to Case 2.

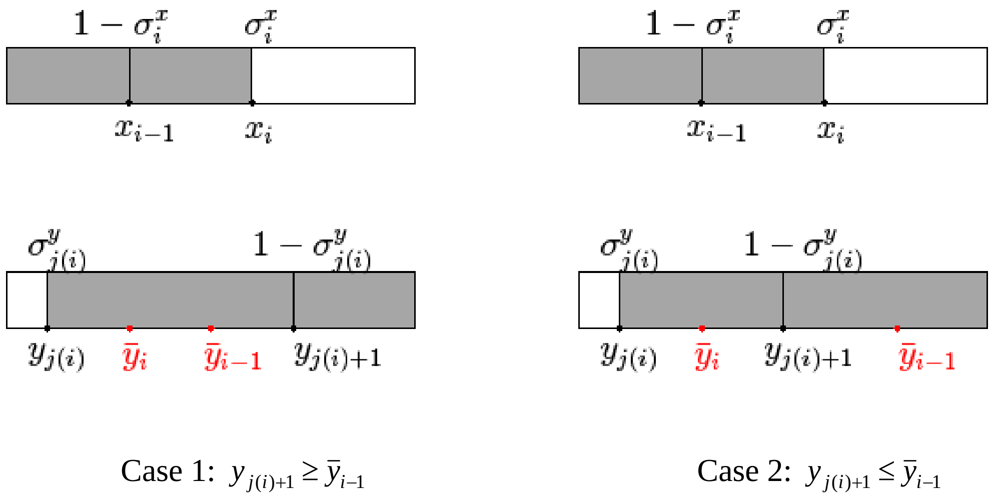


Figure A6.4: Scenarios when upper bound on *X* induces a lower bound on *Y*

i) Scenario 1: Select *X* segment with index *i* and *Y* segment with index $j(i)+1$

This implies $T^{X}_{i+1} = 0$ and $T^{Y}_{j(i)+2} = 0$, which gives $\tau^{Y}_{j(i)+1} \geq (1-q_i)(1-0) - \tau^{X}_{i}\left(p_i - q_i\right) + q_i(0)$, or $\tau^{Y}_{j(i)+1} \geq (1-q_i) - \tau^{X}_{i}\left(p_i - q_i\right)$, which is what we derived in Equation A2.2.4.

ii) Scenario 2: Select *X* segment *i* and *Y* segment with index $j(i)+2$ or higher
This implies $T^{X}_{i+1} = 0$ and $T^{Y}_{j(i)+2} = 1$, which gives $\tau^{Y}_{j(i)+1} \geq (1-q_i)(1-0) - \tau^{X}_{i}\left(p_i - q_i\right) + q_i(1)$, or $\tau^{Y}_{j(i)+1} \geq (1-q_i) - \tau^{X}_{i}\left(p_i - q_i\right) + q_i$, or $\tau^{Y}_{j(i)+1} \geq 1 - \tau^{X}_{i}\left(p_i - q_i\right)$, which is valid because $\tau^{Y}_{j(i)+1}$ is equal to one (because we selected *Y* segment with index $j(i)+2$ or higher) and $1 - \tau^{X}_{i}\left(p_i - q_i\right)$ lies between zero and one because $p_i > q_i$.

iii) Scenario 3: Select *X* segment with index $i-1$ or lower and *Y* segment $j(i)+1$
This implies $T^{X}_{i+1} = 0$, $\tau^{X}_{i} = 0$ and $T^{Y}_{j(i)+2} = 0$, which gives
$\tau^{Y}_{j(i)+1} \geq (1-q_i)(1-0) - 0\left(p_i - q_i\right) + q_i(0)$,
or $\tau^{Y}_{j(i)+1} \geq (1-q_i)$, or $q_i \geq 1 - \tau^{Y}_{j(i)+1}$, or $q_i \geq \sigma^{Y}_{j(i)}$ or $\sigma^{Y}_{j(i)} \leq \dfrac{y_{j(i)+1} - \bar{y}_{i-1}}{y_{j(i)+1} - y_{j(i)}}$ which is the bound on $\sigma^{Y}_{j(i)}$ by considering an upper bound of x_{i-1} on the *X* variable. This scenario is valid only for Case 1 above.

iv) Scenario 4: Select *X* segment with index $i-1$ or lower and *Y* segment with index $j(i)+2$ or higher
This implies $T^{X}_{i+1} = 0$, $\tau^{X}_{i} = 0$ and $T^{Y}_{j(i)+2} = 1$, which gives
$\tau^{Y}_{j(i)+1} \geq (1-q_i)(1-0) - 0\left(p_i - q_i\right) + q_i(1)$, or $\tau^{Y}_{j(i)+1} \geq 1$ which is valid since $\tau^{Y}_{j(i)+1}$ takes a value of zero since we choose a *Y* segment with index $j(i)+2$ or higher.

3. Lower bound on *X* induces an upper bound on *Y*

Consider the situation where *X* and *Y* are related by the constraint $Y \leq -aX + b$. In this situation, a lower bound on *X* induces an upper bound on *Y*. Let us assume that we select an *X* segment with index i+1 or lower. This implies that *X* has a lower bound of x_i. Let $j(i) = \min\ j : -ax_i + b \geq y_j$ .Then, we cannot select a *Y* segment whose index is greater than $j(i)$. This gives us the following valid inequalities.

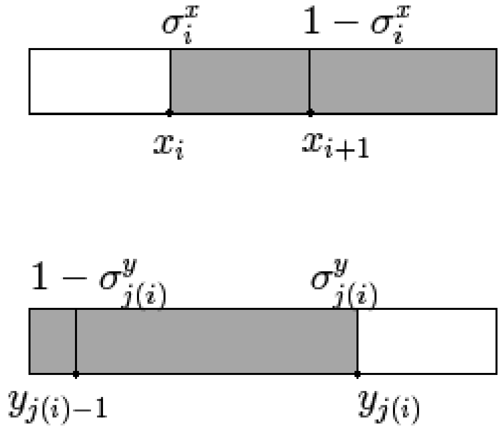


Figure A6.5: Lower bound on *X* inducing an upper bound on *Y*

a) Inequalities relating binary variables

Let $\bar{I}_3$ be the set of break points in the *X* partition such that $I_3 = \{i : 1 \leq j(i) \leq n_Y - 1\}$. Then, if we select an *X* segment with index $i+1$ or higher, then we must select a *Y* segment with index $j(i)$ or lower. So, we have the following inequality.

$$1 - T_{j(i)+1}^Y \geq T_{i+1}^X \text{ for every } i \in I_3 \tag{A6.9}$$

b) Inequalities relating marginal weight variables

Let $\bar{y}_i = -ax_i + b$. Then, if *X* has a lower bound of x_i, *Y* has an upper bound of $\bar{y}_i$. This implies that there is an upper bound on the marginal weight variable for break point *j*(*i*) along the *Y* partition, and consequently an upper bound on the cumulative marginal weight for break point $j(i)$ along the *Y* partition. Consider a case where we want to choose

- segment *i*+1 for the *X* variable with weights σ_i^X and $1-\sigma_i^X$ assigned to break points *i* and *i*+1, and
- segment $j(i)$ for the *Y* variable with weights $\sigma_{j(i)}^Y$ and $1-\sigma_{j(i)}^Y$ assigned to break points $j(i)$ and $j(i)-1$.

Since $Y \le -aX + b$, we get the following.

$$\sigma_{j(i)}^Y y_{j(i)} + (1-\sigma_{j(i)}^Y) y_{j(i)-1} \le -a(\sigma_i^X x_i + (1-\sigma_i^X) x_{i+1}) + b \qquad \text{(A6.10)}$$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) + y_{j(i)-1} \le \sigma_i^X (-ax_i + ax_{i+1}) + (-ax_{i+1} + b)$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) + y_{j(i)-1} \le \sigma_i^X (-ax_i + ax_{i+1}) + \bar{y}_{i+1}$ where $\bar{y}_{i+1} = -ax_{i+1} + b$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) \le \sigma_i^X (-ax_i + ax_{i+1}) + \bar{y}_{i+1} - y_{j(i)-1}$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) \le \sigma_i^X (-ax_i + b - (-ax_{i+1} + b)) + \bar{y}_{i+1} - y_{j(i)-1}$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) \le \sigma_i^X (\bar{y}_i - \bar{y}_{i+1}) + \bar{y}_{i+1} - y_{j(i)-1}$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)-1}) \le \sigma_i^X (\bar{y}_i - y_{j(i)-1} - (\bar{y}_{i+1} - y_{j(i)-1})) + \bar{y}_{i+1} - y_{j(i)-1}$

Or $\quad \sigma_{j(i)}^Y \le \sigma_i^X \left(\dfrac{\bar{y}_i - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}} - \dfrac{\bar{y}_{i+1} - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}} \right) + \dfrac{\bar{y}_{i+1} - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}}$

Or $\quad \sigma_{j(i)}^Y \le \sigma_i^X (p_i - q_i) + q_i$ where $p_i = \dfrac{\bar{y}_i - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}}$ and $q_i = \dfrac{\bar{y}_{i+1} - y_{j(i)-1}}{y_{j(i)} - y_{j(i)-1}}$

Or $\quad \tau_{j(i)}^Y \le (1-\tau_{i+1}^X)(p_i - q_i) + q_i$ since $\sigma_i^X = 1-\tau_{i+1}^X$ and $\sigma_{j(i)}^Y = \tau_{j(i)}^Y$

Or $\quad \tau_{j(i)}^Y \le p_i - \tau_{i+1}^X (p_i - q_i) \qquad \text{(A6.11)}$

Thus, if we select segment $i+1$ for the X variable and assign a cumulative weight of τ_{i+1}^{X} to break point $i+1$, then we have an upper bound on the cumulative weight variable for break point $j(i)$. This leads us to the following inequality.

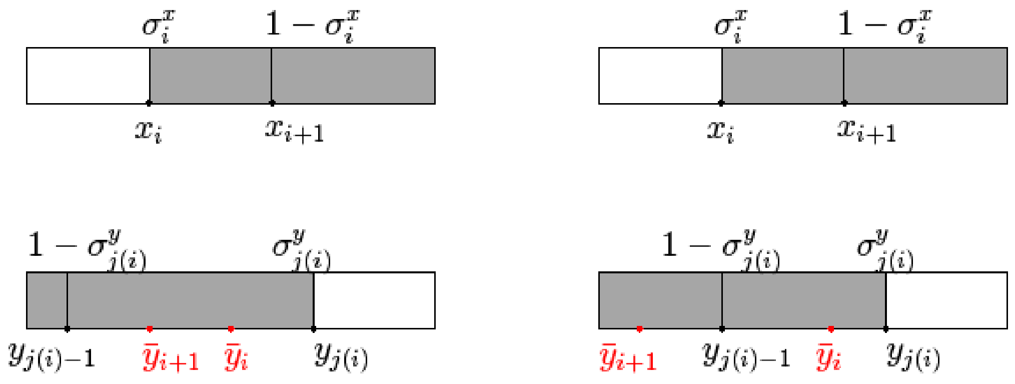
$$\tau_{j(i)}^{Y} \le p_i T_{i+1}^{X} - (p_i - q_i)\tau_{i+1}^{X} - q_i(1 - T_{j(i)}^{Y}) + (1 - T_{i+1}^{X}) \qquad \text{(A6.12)}$$

Proof of validity To show that this inequality is valid, we have to consider two cases.

Case 1: $y_{j(i)-1} \le \bar{y}_{i+1}$

Case 2: $y_{j(i)-1} \ge \bar{y}_{i+1}$

We consider four scenarios that cover these two cases. Scenarios 1, 2, 3 and 4 apply to Case 1 and Scenarios 1, 2 and 4 apply to Case 2.



Case 1: $y_{j(i)-1} \le \bar{y}_{i+1}$ Case 2: $y_{j(i)-1} \ge \bar{y}_{i+1}$

Figure A6.6: Scenarios when lower bound on *X* induces an upper bound on *Y*

i) Scenario 1: Select *X* segment with index $i+1$ and *Y* segment with index $j(i)$

This implies $T_{i+1}^{X}=1$ and $T_{j(i)}^{Y}=1$, which gives $\tau_{j(i)}^{Y} \le p_i-(p_i-q_i)\tau_{i+1}^{X}-q_i(1-1)$, or $\tau_{j(i)}^{Y} \le p_i-(p_i-q_i)\tau_{i+1}^{X}$, which is what we derived in Equation A2.3.4.

ii) Scenario 2: Select *X* segment $i+1$ and *Y* segment with index $j(i)-1$ or lower

This implies $T_{i+1}^{X}=1$ and $T_{j(i)}^{Y}=0$, which gives $\tau_{j(i)}^{Y} \le p_i-(p_i-q_i)\tau_{i+1}^{X}-q_i(1-0)$ or $\tau_{j(i)}^{Y} \le (p_i-q_i)(1-\tau_{i+1}^{X})$., which is valid because $\tau_{j(i)}^{Y}$ is zero (since we selected a *Y* segment with index $j(i)-1$ or lower) and $(p_i-q_i)(1-\tau_{i+1}^{X})$ is positive because $p_i>q_i$ and $0 \le \tau_{i+1}^{X} \le 1$.

iii) Scenario 3: Select *X* segment with index $i+2$ or higher and *Y* segment with indes $j(i)$

This implies $T_{i+1}^{X}=1$, $\tau_{i+1}^{X}=1$ and $T_{j(i)}^{Y}=1$, which gives $\tau_{j(i)}^{Y} \le p_i-(p_i-q_i)-q_i(1-1)$, or $\tau_{j(i)}^{Y} \le q_i$, or $\sigma_{j(i)}^{Y} \le q_i$ or $\sigma_{j(i)}^{Y} \le \dfrac{\bar{y}_{i+1}-y_{j(i)-1}}{y_{j(i)}-y_{j(i)-1}}$ which is the bound on $\sigma_{j(i)}^{Y}$ given that *X* has a lower bound of x_{i+1}. This scenario is valid only for Case 1 above.

iv) Scenario 4: Select *X* segment with index $i+2$ or higher and *Y* segment with index $j(i)-1$ or lower

This implies $T_{i+1}^{X}=1$, $\tau_{i+1}^{X}=1$ and $T_{j(i)}^{Y}=0$, which gives $\tau_{j(i)}^{Y} \le p_i-(p_i-q_i)-q_i(1-0)$, or $\tau_{j(i)}^{Y} \le 0$ which is valid since $\tau_{j(i)}^{Y}$ is equal to zero as we select a *Y* segment with index $j(i)-1$ or .

4. Lower bound on *X* induces a lower bound on *Y*

Consider the situation where *X* and *Y* are related by the constraint $Y \geq aX + b$. In this situation, a lower bound on *X* induces a lower bound on *Y*. Let us assume that we select an *X* segment with index $i+1$ or higher. This implies that *X* cannot be lower than x_i. Let $j(i) = \max j : ax_i + b \leq y_j$. Then, we cannot select a *Y* segment whose index is less than $j(i)+1$. This gives us the following valid inequalities.

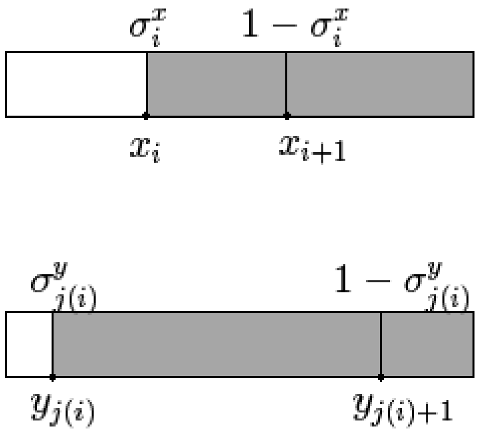


Figure A6.7: Lower bound on *X* inducing a lower bound on *Y*

a) Inequalities relating binary variables

Let $\bar{I}_4$ be the set of break points in the *X* pattern such that $I_4 = \{i : 1 \leq j(i) \leq n_Y - 1\}$. Then, if we select an *X* segment with index $i+1$ or higher, then we must select a *Y* segment with index $j(i)+1$ or higher. So, we have the following inequality.

$$T_{j(i)+1}^{Y} \geq T_{i+1}^{X} \quad \text{for every } i \in I_1 \tag{A6.13}$$

b) Inequalities relating marginal weight variables

Let $\bar{y}_i = ax_i + b$. Then, if X has a lower bound of x_i, *Y* has a lower bound of $\bar{y}_i$. This implies that there is an upper bound on the marginal weight variable for break point *j*(*i*) along the *Y* partition and consequently a lower bound on the cumulative marginal weight for break point $j(i)+1$ along the *Y* partition. Consider a case where we want to choose

- segment $i+1$ for the *X* variable with weights σ_i^X and $1-\sigma_i^X$ assigned to break points *i* and *i*+1
- segment $j(i)+1$ for the *Y* variable with weights $\sigma_{j(i)}^Y$ and $1-\sigma_{j(i)}^Y$ assigned to break points $j(i)$ and $j(i)+1$.

Since $Y \geq aX + b$, we get the following.

$$\sigma_{j(i)}^Y y_{j(i)} + (1-\sigma_{j(i)}^Y) y_{j(i)+1} \geq a(\sigma_i^X x_i + (1-\sigma_i^X) x_{i+1}) + b \qquad \text{(A6.14)}$$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)+1}) + y_{j(i)+1} \geq a\sigma_i^X x_i - a\sigma_i^X x_{i+1} + ax_{i+1} + b$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)+1}) + y_{j(i)+1} \geq \sigma_i^X (ax_i - ax_{i+1}) + \bar{y}_{i+1}$ where $\bar{y}_{i+1} = ax_i + b$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)} - y_{j(i)+1}) \geq \sigma_i^X (ax_i - ax_{i+1}) + \bar{y}_{i+1} - y_{j(i)+1}$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)+1} - y_{j(i)}) \leq \sigma_i^X (ax_{i+1} - ax_i) + y_{j(i)+1} - \bar{y}_{i+1}$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)+1} - y_{j(i)}) \leq \sigma_i^X (ax_{i+1} + b - (ax_i + b)) + y_{j(i)+1} - \bar{y}_{i+1}$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)+1} - y_{j(i)}) \leq \sigma_i^X (\bar{y}_{i+1} - \bar{y}_i) + y_{j(i)+1} - \bar{y}_{i+1}$ where $\bar{y}_i = ax_i + b$

Or $\quad \sigma_{j(i)}^Y (y_{j(i)+1} - y_{j(i)}) \leq \sigma_i^X (-(y_{j(i)+1} - \bar{y}_{i+1}) + (y_{j(i)+1} - \bar{y}_i)) + y_{j(i)+1} - \bar{y}_{i+1}$

Or $\quad \sigma_{j(i)}^Y \leq \sigma_i^X \left(-\frac{y_{j(i)+1} - \bar{y}_{i+1}}{y_{j(i)+1} - y_{j(i)}} + \frac{y_{j(i)+1} - \bar{y}_i}{y_{j(i)+1} - y_{j(i)}} \right) + \frac{y_{j(i)+1} - \bar{y}_{i+1}}{y_{j(i)+1} - y_{j(i)}}$ since $y_{j(i)+1} > y_{j(i)}$

Or $\quad \sigma_{j(i)}^Y \leq \sigma_i^X (p_i - q_i) + q_i$ where $p_i = \frac{y_{j(i)+1} - \bar{y}_i}{y_{j(i)+1} - y_{j(i)}}$ and $q_i = \frac{y_{j(i)+1} - \bar{y}_{i+1}}{y_{j(i)+1} - y_{j(i)}}$

Or $\quad 1-\tau_{j(i)+1}^Y \leq (1-\tau_{i+1}^X)(p_i - q_i) + q_i$ since in this case $\sigma_i^X = 1-\tau_{i+1}^X$ and $\sigma_{j(i)}^Y = 1-\tau_{j(i)+1}^Y$

Or $\quad 1-\tau_{j(i)+1}^Y \leq p_i - q_i - \tau_{i+1}^X (p_i - q_i) + q_i$

Or $\tau^Y_{j(i)+1} \geq (1-p_i) + (p_i - q_i)\tau^X_{i+1}$ (A6.15)

Thus, if we select segment $i+1$ for the X variable and assign a cumulative weight of τ^X_{i+1} to break point $i+1$, then we have an upper bound on the cumulative weight variable for break point $j(i)+1$ along the Y partition. This leads us to the following inequality.

$$\tau^Y_{j(i)+1} \geq (p_i - q_i)\tau^X_{i+1} + (1-p_i)T^X_{i+1} + q_i T^Y_{j(i)+2} \qquad \text{(A6.16)}$$

Proof of validity To show that this inequality is valid, we have to consider two cases.

Case 1: $y_{j(i)+1} \geq \bar{y}_{i-1}$

Case 2: $y_{j(i)+1} \leq \bar{y}_{i-1}$

We consider four scenarios that cover these two cases. Scenarios 1, 2, 3 and 4 apply to Case 1 and Scenarios 1, 2 and 4 apply to Case 2.

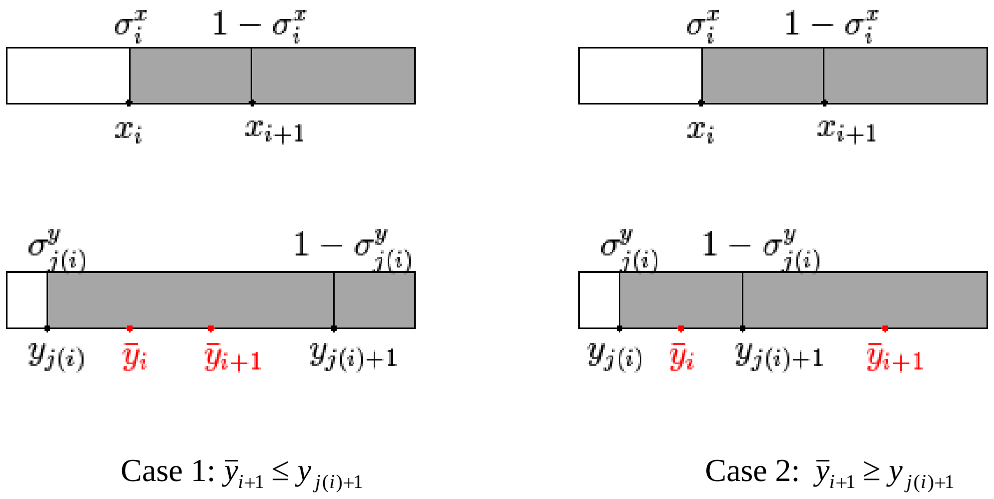


Figure A6.8: Scenarios when lower bound on X induces a lower bound on Y

i) Scenario 1: Select *X* segment $i+1$ and *Y* segment $j(i)+1$

This implies $T^X_{i+1}=1$ and $T^Y_{j(i)+2}=0$, therefore $\tau^Y_{j(i)+1} \geq (p_i - q_i)\tau^X_{i+1} + (1-p_i) + q_i(0)$, or $\tau^Y_{j(i)+1} \geq (p_i - q_i)\tau^X_{i+1} + (1-p_i)$, which is what we derived in Equation (A2.2.4).

ii) Scenario 2: Select *X* segment $i+1$ and *Y* segment with index $j(i)+2$ or higher

This implies $T^X_{i+1}=1$ and $T^Y_{j(i)+2}=1$, therefore $\tau^Y_{j(i)+1} \geq (p_i - q_i)\tau^X_{i+1} + (1-p_i) + q_i$ or $\tau^Y_{j(i)+1} \geq 1-(p_i - q_i)(1-\tau^X_{i+1})$, which is valid because $\tau^Y_{j(i)+1}$ is equal to one (since we select a *Y* segment with index $j(i)+2$ or higher) and $1-(p_i - q_i)(1-\tau^X_{i+1})$ is less than 1 since $p_i \geq q_i$ and $0 \leq \tau^X_{i+1} \leq 1$.

iii) Scenario 3: Select *X* segment with index $i+2$ or higher and *Y* segment with index $j(i)+1$

This implies $T^X_{i+1}=1$, $\tau^X_{i+1}=1$ and $T^Y_{j(i)+2}=0$, therefore $\tau^Y_{j(i)+1} \geq (p_i - q_i)(1) + (1-p_i) + q_i(0)$ or $\tau^Y_{j(i)+1} \geq 1-q_i$ or $1-\tau^Y_{j(i)+1} \geq q_i$ or $\sigma^Y_{j(i)} \leq q_i$ or $\sigma^Y_{j(i)} \leq \dfrac{y_{j(i)+1} - \bar{y}_{i+1}}{y_{j(i)+1} - y_{j(i)}}$ which is the upper bound on $\sigma^Y_{j(i)}$ given that *X* has a lower bound of x_{i-1}. This scenario is only applicable for the case when $\bar{y}_{i+1} \leq y_{j(i)+1}$.

iv) Scenario 4: Select *X* segment with index $i+2$ or higher and *Y* segment with index $j(i)+2$ or higher

This implies $T^X_{i+1}=1$, $\tau^X_{i+1}=1$ and $T^Y_{j(i)+2}=1$, therefore $\tau^Y_{j(i)+1} \geq (p_i - q_i) + (1-p_i) + q_i$ or $\tau^Y_{j(i)+1} \geq 1$, which is valid because $\tau^Y_{j(i)+1}$ takes a value of one (since we select *Y* segment with index $j(i)+2$ or higher).

Appendix A7: Selected Functional Decompositions of Ali et al. Problems

In this appendix, we describe how we decomposed the non-linear functions in some of the test problems that we used in our study.

1) Ackleys

$$\text{Min}\, z = -20\exp\left(-0.02\sqrt{n^{-1}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(n^{-1}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \qquad \text{(A7.1)}$$

$$\text{s.t.} -30 \le x_i \le 30,\ i \in \{1,2,\ldots,n\}$$

Reformulation:

$$\text{Min}\, z = -20u - v + 20 + e \qquad \text{(A7.1.2)}$$

$$\text{subject to } y = \sum_{i=1}^{n} y_i \qquad \text{(A7.1.3)}$$

$$w = \sum_{i=1}^{n} w_i \qquad \text{(A7.1.4)}$$

$$y_i = x_i^2,\ i \in \{1,2,\ldots,n\} \qquad \text{(A7.1.5)}$$

$$w_i = \cos(2\pi x_i),\ i \in \{1,2,\ldots,n\} \qquad \text{(A7.1.6)}$$

$$u = \exp\left(-0.02\sqrt{n^{-1}y}\right) \qquad \text{(A7.1.7)}$$

$$v = \exp\left(n^{-1}w\right) \qquad \text{(A7.1.8)}$$

We used one-dimensional grids to linearize Equations (A7.1.5 –A7.1.8). Also, n =10.

2) AluffiPentini

$$\text{Min}\, z = 0.25x_1^4 - 0.5x_1^2 + 0.1x_1 + 0.5x_2^2 \qquad \text{(A7.2.1)}$$

$$s.t. -10 \le x_i \le 10,\ i \in \{1,2\}$$

Reformulation

$$\text{Min}\, z = y_1 + y_2 \qquad \text{(A7.2.2)}$$

$$\text{s.t. } y_1 = 0.25x_1^4 - 0.5x_1^2 + 0.1x_1 \qquad \text{(A7.2.3)}$$

$$y_2 = 0.5x_2^2 \tag{A7.2.4}$$

We used one dimensional grids to linearize Equations (7.2.2 – 7.2.4).

3) Becker and Lago Problem

$$\text{Min}\, z = (|x_1| - 5)^2 + (|x_2| - 5)^2 \tag{A7.3.1}$$

$$s.t. -10 \le x_i \le 10,\ i \in \{1, 2\}$$

<u>Reformulation</u>

$$\text{Min}\, z = y_1 + y_2 \tag{A7.3.2}$$

$$\text{s.t.}\ \ y_1 = (|x_1| - 5)^2 \tag{A7.3.3}$$

$$\mathrm{y}_2 = (|x_2| - 5)^2 \tag{A7.3.4}$$

We used one dimensional grids to linearize Equations (7.3.2 – 7.3.4).

4) Bohachevsky I

$$\text{Min}\, z = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7 \tag{A7.4.1}$$

$$s.t. -50 \le x_i \le 50,\ i \in \{1, 2\}$$

<u>Reformulation</u>

$$\text{Min}\, z = y_1 + y_2 + 0.7 \tag{A7.4.2}$$

$$\text{s.t.}\ \ \mathrm{y}_1 = x_1^2 - 0.3\cos(3\pi x_1) \tag{A7.4.3}$$

$$y_2 = 2x_2^2 - 0.4\cos(4\pi x_2) \tag{A7.4.4}$$

We used one dimensional grids to linearize Equations (7.4.2 – 7.4.4).

5) Bohachevsky II

$$\text{Min}\, z = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3 \tag{A7.5.1}$$

$$s.t. -50 \le x_i \le 50,\ i \in \{1, 2\}$$

We used a single two-dimensional grid to linearize Equation (7.5.1).

6) Branin

$$\mathrm{Min}\, z = a\left(x_2 - bx_1^2 + cx_1 - d\right)^2 + g(1-h)\cos(x_1) + g \tag{A7.6.1}$$

$$s.t. -5 \le x_1 \le 10,\ \ 0 \le x_2 \le 15$$

where $a = 1, b = {5.1}/{4\pi^2}, c = {5}/{\pi}, d = 6, g = 10, h = {1}/{8\pi}$

We used a single two-dimensional grid to linearize Equation (7.6.1).

7) Camel Back 3

$$\mathrm{Min}\, z = 2x_1^2 - 1.05x_1^4 + 1.6x_1^6 + x_1x_2 + x_2^2 \tag{A7.7.1}$$

$$s.t. -5 \le x_i \le 5,\ i \in \{1, 2\}$$

We used a single two-dimensional grid to linearize Equation (7.7.1).

8) Camel Back 6

$$\mathrm{Min}\, z = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \tag{A7.8.1}$$

$$s.t. -5 \le x_i \le 5,\ i \in \{1, 2\}$$

We used a single two-dimensional grid to linearize Equation (7.8.1).

9) Cosine Mixture Problem

$$\mathrm{Min}\, z = 0.1\sum_{i=1}^{n}\cos(5\pi x_i) - \sum_{i=1}^{n} x_i^2 \tag{A7.9.1}$$

$$\mathrm{s.}t. -1 \le x_i \le 1,\ i \in \{1, 2, ..., n\}$$

<u>Reformulation</u>

$$\mathrm{Min}\, z = \sum_{i=1}^{n} y_i \tag{A7.9.2}$$

$$\text{s.t. } \mathrm{y_i} = 0.1\cos(5\pi x_i) - x_i^2,\ i \in \{1, 2, ..., n\} \tag{A7.9.3}$$

We used one-dimensional grids to linearize Equations (7.9.3). Also, *n* = 2, 4.

10) Dekkers and Aarts Problem

$$\mathrm{Min}\, z = 10^5 x_1^2 + x_2^2 - \left(x_1^2 + x_2^2\right)^2 x_1x_2 + 10^{-5}\left(x_1^2 + x_2^2\right)^4 \tag{A7.10.1}$$

$s.t. -20 \le x_i \le 20,\ i \in \{1,2\}$

We used a single two-dimensional grid to linearize Equation (7.10.1).

11) Easom Problem

$$\text{Min}\, z = -\cos(x_1)\cos(x_2)\exp\left(-(x_1-\pi)^2-(x_2-\pi)^2\right) \quad \text{(A7.11.1)}$$

$s.t. -10 \le x_i \le 10,\ i \in \{1,2\}$

We used a single two-dimensional grid to linearize Equation (7.11.1).

12) Epistatic Michalewicz Problem

$$\text{Min}\, z = -\sum_{i=1}^{n}\cos(y_i)\left(\sin\left(\frac{iy_i^2}{\pi}\right)\right)^{2m} \quad \text{(A7.12.1)}$$

$$s.t.\ y_i = \begin{cases} x_i\cos(\theta)-x_{i+1}\sin(\theta),\ i=1,3,5...,<n \\ x_i\cos(\theta)+x_{i-1}\sin(\theta),\ i=2,4,6...,<n \\ x_i,\ i=n \end{cases} \quad \text{(A7.12.2)}$$

$0 \le x_i \le \pi,\ i \in \{1,\ 2,...,n\},\ \theta = \pi/6,\ m=10$

We used a single one-dimensional grid to linearize Equation (7.12.1). Also, *n* was taken as 5.

13) Exponential Problem

$$\text{Max}\, z = \exp\left(-0.5\sum_{i=1}^{n}x_i^2\right) \quad \text{(A7.13.1)}$$

$s.t. -1 \le x_i \le 1,\ i \in \{1,\ 2,...,n\}$

Reformulation:

$$\text{Min}\, z = -u \quad \text{(A7.13.2)}$$

subject to $u = \exp(-0.5y)$ (A7.13.3)

$$y = \sum_{i=1}^{n} y_i \tag{A7.13.4}$$

$$y_i = x_i^2,\ i \in \{1, 2, ..., n\} \tag{A7.13.5}$$

We used one-dimensional grids to linearize Equation (7.13.3) and (7.13.5). Also, *n* was taken as 10.

14) Goldstein and Price

$$\begin{aligned} \text{Min}\, z = &\left(1 + \left(x_1 + x_2 + 1\right)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right) \\ &* \\ &\left(30 + \left(2x_1 - 3x_2\right)^2 \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right) \end{aligned} \tag{A7.14.1}$$

$$s.t. -2 \le x_i \le 2,\ i \in \{1, 2\}$$

We used a single two-dimensional grid to linearize Equation (7.14.1).

15) Griewank Problem

$$\text{Min}\, z = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{A7.15.1}$$

$$s.t. -2 \le x_i \le 2,\ i \in \{1, 2\}$$

<u>Reformulation (n=10)</u>

$$\text{Min}\, z = 1 + \frac{1}{4000}\sum_{i=1}^{n} w_i - y_n \tag{A7.15.2}$$

$$\text{s.t. } w_i = x_i^2,\ i \in \{1, 2, ..., n\} \tag{A7.15.3}$$

$$y_2 = \cos\left(\frac{x_1}{\sqrt{1}}\right)\cos\left(\frac{x_2}{\sqrt{2}}\right) \tag{A7.15.4}$$

$$y_i = y_{i-1}\cos\left(\frac{x_i}{\sqrt{i}}\right),\ i \in \{3, ..., n\} \tag{A7.15.5}$$

We used a one-dimensional grids to linearize Equation (7.15.3) and two-dimensional grids to linearize Equations (7.15.4 –7.15.5).

16) Gulf Research Problem

$$\text{Min}\, z = \sum_{i=1}^{99}\left[\exp\left(-\frac{(u_i - x_2)^{x_3}}{x_1}\right) - 0.01i\right]^2 \tag{A7.16.1}$$

$$\text{where}\, u_i = 25 + \left[-50\ln(0.01i)\right]^{1/1.5}$$

$$0.1 \le x_1 \le 100,\quad 0 \le x_2 \le 25.6,\quad 0 \le x_3 \le 5$$

We used a single three-dimensional grid to linearize Equations (7.16.1).

17) Hartman 3

$$\text{Min}\, z = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij}\left(x_j - p_{ij}\right)^2\right) \tag{A7.17.1}$$

$s.t.\ 0 \le x_j \le 1,\ j \in \{1,2,3\}$, constants a_{ij}, p_{ij} and c_i as in Table 7.1.

i	c_i	a_{ij}			p_{ij}		
		j=1	*j*=2	*j*=3	*j*=1	*j*=2	*j*=3
1	1	3	10	30	0.3689	0.117	0.2673
2	1.2	0.1	10	35	0.4699	0.4387	0.747
3	3	3	10	30	0.1091	0.8732	0.5547
4	3.2	0.1	10	35	0.03815	0.5743	0.8828

Table A7.1: Data for Hartman 3

Reformulation:

$$\text{Min}\, z = -\sum_{i=1}^{4} c_i u_i \tag{A7.17.2}$$

s.t. $$w_i = \sum_{j=1}^{3} y_{ij}\,,\ i \in \{1,2,3,4\} \tag{A7.17.3}$$

$y_{ij} = a_{ij}\left(x_j - p_{ij}\right)^2,\ i \in \{1,2,3,4\},\ j \in \{1,2,3\}$ (A7.17.4)

$u_i = \exp\left(-w_i\right), \quad i \in \{1,2,3,4\}$ (A7.17.5)

We used one-dimensional grids to linearize Equations (A7.17.4–A 7.17.5).

18) Hartman 6

$$\mathrm{Min}\, z = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} a_{ij}\left(x_j - p_{ij}\right)^2\right) \quad \text{(A7.18.1)}$$

$s.t.\ 0 \le x_j \le 1,\ j \in \{1,..,6\}$ and constants a_{ij} , p_{ij} and c_j given in Tables 7.2 and 7.3.

i	c_i	a_{ij}					
		j=1	*j*=2	*j*=3	*j*=4	*j*=5	*j*=6
1	1	10	3	17	3.5	1.7	8
2	1.2	0.05	10	17	0.1	8	14
3	3	3	3.5	1.7	10	17	8
4	3.2	17	8	0.05	10	0.1	14

Table A7.2: Parameter set I for Hartman 6

i	p_{ij}					
	j=1	*j*=2	*j*=3	*j*=4	*j*=5	*j*=6
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.665
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

Table A7.3: Parameter set II for Hartman 6

<u>Reformulation</u>

$$\text{Min}\, z = -\sum_{i=1}^{4} c_i u_i \qquad \text{(A7.18.2)}$$

$$w_i = \sum_{j=1}^{6} y_{ij}\,, \quad i \in \{1,2,3,4\} \qquad \text{(A7.18.3)}$$

$$y_{ij} = a_{ij}\left(x_j - p_{ij}\right)^2, \; i \in \{1,2,3,4\}, \; j \in \{1,..,6\} \qquad \text{(A7.18.4)}$$

$$u_i = \exp\left(-w_i\right), \quad i \in \{1,2,3,4\} \qquad \text{(A7.18.5)}$$

We used one-dimensional grids to linearize Equations (A7.18.4–A7.18.5).

19) Helical Valley

$$\text{Min}\, z = 100\left[\left(x_2 - 10\theta\right)^2 + \left(\sqrt{x_1^2 + x_2^2} - 1\right)^2\right] + x_3^2 \qquad \text{(A7.19.1)}$$

$$\text{where}\, \theta = \begin{cases} \frac{1}{2\pi}\tan^{-1}\left(x_2 / x_1\right), & if \;\; x_1 \ge 0 \\ \frac{1}{2\pi}\tan^{-1}\left(x_2 / x_1\right) + \frac{1}{2}, & if \;\; x_1 < 0 \end{cases}$$

$$-10 \le x_i \le 10, \; i \in \{1,..,3\}$$

<u>Reformulation:</u>

$$\text{Min}\, z = w_{12} + w_3 \qquad \text{(A7.19.2)}$$

subject to

$$w_{12} = 100\left[\left(x_2 - 10\theta\right)^2 + \left(\sqrt{x_1^2 + x_2^2} - 1\right)^2\right] + x_3^2$$

$$\text{where}\, \theta = \begin{cases} \frac{1}{2\pi}\tan^{-1}\left(x_2 / x_1\right), & if \;\; x_1 \ge 0 \\ \frac{1}{2\pi}\tan^{-1}\left(x_2 / x_1\right) + \frac{1}{2}, & if \;\; x_1 < 0 \end{cases} \qquad \text{(A7.19.3)}$$

$$w_3 = x_3^2 \qquad \text{(A7.19.4)}$$

We used a one-dimensional grid for Equations (A7.19.4) and a two dimensional grid for Equation (A7.19.3).

20) Hosaki

$$\mathrm{Min}\, z = \left[1 - 8x_1 + 7x_1^2 - \left(\tfrac{7}{3}\right)x_1^3 + \left(\tfrac{1}{4}\right)x_1^4\right]\left[x_2^2\right]\left[\exp(-x_2)\right] \tag{A7.20.1}$$

$s.t.\, 0 \le x_1 \le 5,\ 0 \le x_2 \le 6$

We used a single two-dimensional grid to linearize Equation (A7.20.1).

21) Kowalik

$$\mathrm{Min}\, z = \sum_{i=1}^{11}\left(a_i - \frac{x_1(1 + x_2 b_i)}{(1 + x_3 b_i + x_4 b_i^2)}\right)^2 \tag{A7.21.1}$$

$s.t.\ 0 \le x_i \le 0.42,\ i \in \{1,..,4\}$ and the values of a_i and b_i as given in Table 7.4.

i	a_i	b_i
1	0.1957	0.25
2	0.1947	0.50
3	0.1735	1.0
4	0.16	2.0
5	0.0884	4.0
6	0.0627	6.0
7	0.0456	8.0
8	0.0342	10.0
9	0.0323	12.0
10	0.0235	14.0
11	0.0246	16.0

Table A7.4: Data for the Kowalik Problem

We used a four-dimensional grid to linearize Equations (A7.21.1).

22) Levy and Montalvo I

$$\mathrm{Min}\, z = \left(\frac{\pi}{n}\right)\left(10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}\left[(y_i - 1)^2\left(1 + 10\sin^2(\pi y_{i+1})\right)\right] + (y_n - 1)^2\right) \qquad \text{(A7.22.1)}$$

where $y_i = 1 + 0.25(x_i + 1)$

$-10 \le x_i \le 10,\ i \in \{1,..,n\}$

<u>Reformulation</u>

$$\mathrm{Min}\, z = \left(\frac{\pi}{n}\right)\sum_{i=0}^{n} u_i \qquad \text{(A7.22.2)}$$

$$u_0 = 10\sin^2(\pi y_1) \qquad \text{(A7.22.3)}$$

$$u_i = (y_i - 1)^2\left(1 + 10\sin^2(\pi y_{i+1})\right),\ i \in \{1,..,n-1\} \qquad \text{(A7.22.4)}$$

$$u_n = (y_n - 1)^2 \qquad \text{(A7.22.5)}$$

$$y_i = 1 + 0.25(x_i + 1), i \in \{1,..,n\} \qquad \text{(A7.22.6)}$$

We used one-dimensional grids to linearize Equations (A7.22.3) and Equations (A7.22.5), and two-dimensional grids to linearize Equation (A7.22.4).

23) Levi and Montalvo 2

$$\mathrm{Min}\, z = 0.1\left(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}\left[(x_i - 1)^2\left(1 + \sin^2(3\pi x_{i+1})\right)\right] + (x_n - 1)^2\left(1 + \sin^2(2\pi x_n)\right)\right)$$

(A7.23.1)

$s.t. -5 \le x_i \le 5,\ i \in \{1,..,n\}$

<u>Reformulation</u>:

$$\mathrm{Min}\, z = 0.1\sum_{i=0}^{n} u_i \qquad \text{(A7.23.2)}$$

$$u_0 = \sin^2(3\pi x_1) \qquad \text{(A7.23.3)}$$

$$u_i = (x_i - 1)^2\left(1 + \sin^2(3\pi x_{i+1})\right),\ i \in \{1,..,n-1\} \qquad \text{(A7.23.4)}$$

$$u_n = (x_n - 1)^2 \left(1 + \sin^2(2\pi x_n)\right) \qquad \text{(A7.23.5)}$$

We used one-dimensional grids to linearize Equations (A7.23.3) and (A7.23.5), and two-dimensional grids to linearize Equations (A7.23.4).

24) McCormick Problem

$$\mathrm{Min}\, z = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1 \qquad \text{(A7.24.1)}$$

$$s.t. -1.5 \le x_1 \le 4,\ -3 \le x_2 \le 3$$

We used one-dimensional grids to linearize Equations (A7.24.1).

25) Meyer and Roth Problem

$$\mathrm{Min}\, z = \sum_{i=1}^{5} \left(\frac{x_1 x_3 t_i}{(1 + x_1 t_i + x_2 v_i)} - y_i \right)^2 \qquad \text{(A7.25.1)}$$

$$s.t.\ -10 \le x_j \le 10,\quad j \in \{1,..,3\}$$

The values of the parameters t_i, v_i and y_i are given in Table 7.5.

i	t_i	v_i	y_i
1	1.0	1.0	0.126
2	2.0	1.0	0.219
3	1.0	2.0	0.076
4	2.0	2.0	0.126
5	0.1	0.0	0.186

Table A7.5: Data for Meyer and Roth Problem

Reformulation

We used a three-dimensional grid to linearize Equations (A7.25.1).

26) Miele and Cantrell Problem

$$\text{Min}\, z = \left(\exp(x_1) - x_2\right)^4 + 100(x_2 - x_3)^6 + \left(\tan(x_3 - x_4)\right)^4 + x_1^8 \qquad \text{(A7.26.1)}$$

$s.t.\ -1 \le x_i \le 1,\ i \in \{1,..,4\}$

<u>Reformulation</u>

$$\text{Min}\, z = y_1 + y_2 + y_3 + y_4 \qquad \text{(A7.26.2)}$$

$$y_1 = \left(\exp(x_1) - x_2\right)^4 \qquad \text{(A7.26.3)}$$

$$\text{y}_2 = 100(x_2 - x_3)^6 \qquad \text{(A7.26.4)}$$

$$\text{y}_3 = \left(\tan(x_3 - x_4)\right)^4 \qquad \text{(A7.26.5)}$$

$$y_4 = x_1^8 \qquad \text{(A7.26.6)}$$

We used a one-dimensional grid to linearize Equations (A7.26.6) and two-dimensional grids to linearize equations (A7.26.3–A7.26.5).

27) Modified Langerman Problem

$$\text{Min}\, z = -\sum_{j=1}^{5} c_j \cos\left(\frac{d_j}{\pi}\right) \exp\left(-\pi d_j\right) \qquad \text{(A7.27.1)}$$

$$\text{where}\, d_j = \sum_{i=1}^{n} \left(x_i - a_{ji}\right)^2 \qquad \text{(A7.27.2)}$$

$0 \le x_i \le 10,\ i \in \{1,..,n\}$

The values of the parameters c_j and a_{ji} are given in Tables 7.6 and 7.7.

	a_{ji}										c_j
j	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10	
1	9.681	0.667	4.783	9.095	3.517	9.325	6.544	0.211	5.122	2.02	0.806
2	9.4	2.041	3.788	7.931	2.882	2.672	3.568	1.284	7.033	7.374	0.517
3	8.025	9.152	5.114	7.621	4.564	4.711	2.996	6.126	0.734	4.982	0.1
4	2.196	0.415	5.649	6.979	9.51	9.166	6.304	6.054	9.377	1.426	0.908
5	8.074	8.777	3.467	1.867	6.708	6.349	4.534	0.276	7.633	1.567	0.965

Table A7.6: Data for Modified Langerman Problem

Reformulation

$$\mathrm{Min}\, z = \sum_{i=1}^{5} u_j \tag{A7.27.3}$$

$$w_{ji} = \left(x_i - a_{ji}\right)^2, \quad j \in \{1,..,5\},\, i \in \{1,..,n\} \tag{A7.27.4}$$

$$d_j = \sum_{i=1}^{n} w_{ji}, \quad j \in \{1,..,5\} \tag{A7.27.5}$$

$$u_j = c_j \cos\left(\frac{d_j}{\pi}\right) \exp\left(-\pi d_j\right), \quad j \in \{1,..,5\} \tag{A7.27.6}$$

We used one-dimensional grids to linearize Equations (A7.27.4) and Equations (A7.27.6).

28) Modified Rosenbrock Problem

$$\mathrm{Min}\, z = 100\left(x_2 - x_1^2\right)^2 + \left(6.4\left(x_2 - 0.5\right)^2 - x_1 - 0.6\right)^2 \tag{A7.28.1}$$

$$s.t. -5 \le x_i \le 5,\ i \in \{1,2\}$$

We used a two-dimensional grid to linearize Equation (A7.28.1).

29) Multi-Gaussian Problem

$$\mathrm{Max}\, z = \sum_{i=1}^{5} a_i \exp\left(-\frac{\left(x_1 - b_i\right)^2 + \left(x_2 - c_i\right)^2}{\mathrm{d}_\mathrm{i}^2}\right) \tag{A7.29.1}$$

$$s.t. -2 \le x_i \le 2,\ i \in \{1,2\}$$

Reformulation

$$\mathrm{Max}\, z = \sum_{i=1}^{5} y_i \tag{A7.29.2}$$

$$\text{s.t.} \qquad \mathrm{y_i} = a_i \exp\left(-\frac{\left(x_1 - b_i\right)^2 + \left(x_2 - c_i\right)^2}{\mathrm{d}_\mathrm{i}^2}\right), \quad i \in \{1,..,5\} \tag{A7.29.3}$$

We used two-dimensional grids to linearize Equation (A7.29.3).

30) Neumaier 2 Problem

$$\text{Min}\, z = \sum_{k=1}^{n}\left(b_k - \sum_{i=1}^{n}(x_i)^k\right)^2 \tag{A7.30.1}$$

$$s.t.\ 0 \le x_i \le n,\ \ i \in \{1,..,n\}$$

<u>Reformulation</u>

$$\text{Min}\, z = \sum_{k=1}^{n} u_k \tag{A7.30.2}$$

$$y_{ik} = (x_i)^k,\ \ i \in \{1,..,n\},\ k \in \{1,..,n\} \tag{A7.30.3}$$

$$w_k = \sum_{i=1}^{n} y_{ik},\ \ k \in \{1,..,n\} \tag{A7.30.4}$$

$$u_k = (b_k - w_k)^2,\ \ k \in \{1,..,n\} \tag{A7.30.5}$$

We used one-dimensional grids to linearize Equations (A7.30.3) and Equations (A7.30.5), and used n = 4 and b = {8, 18, 44, 114}.

31) Neumaier 3 Problem

$$\text{Min}\, z = \sum_{i=1}^{n}(x_i - 1)^2 - \sum_{i=2}^{n}(x_i x_{i-1}) \tag{A7.31.1}$$

$$s.t.\ -n^2 \le x_i \le n^2,\ \ i \in \{1,..,n\}$$

<u>Reformulation</u>

$$\text{Min}\, z = \sum_{i=1}^{n} y_i - \sum_{i=2}^{n} w_i \tag{A7.31.2}$$

$$y_i = (x_i - 1)^2,\ \ i \in \{1,..,n\} \tag{A7.31.3}$$

$$w_i = x_i x_{i-1},\ \ i \in \{2,..,n\} \tag{A7.31.4}$$

We used one-dimensional grids to linearize Equations 7.31.3 and two-dimensional grids to linearize Equations 7.31.4.

32) Odd Square Problem

$$\text{Min}\, z = -\frac{1.0 + 0.2d}{D + 0.1}\cos(D\pi)\exp\left(-\frac{D}{2\pi}\right) \quad \text{(A7.32.1)}$$

$$d = \sqrt{\sum_{i=1}^{n}(x_i - b_i)^2} \quad \text{(A7.32.2)}$$

$$D = \sqrt{n}\max(x_i - b_i) \quad \text{(A7.32.3)}$$

$$-15 \le x_i \le 15, \;\; i \in \{1,..,20\}$$

$$b = \begin{Bmatrix} 1, 1.3, 0.8, \text{-}0.4, \text{-}1.3, 1.6, \text{-}2, \text{-}6, 0.5, 1.4, \\ 1, 1.3, 0.8, \text{-}4, \text{-}1.3, 1.6, \text{-}0.2, \text{-}0.6, 0.5, 1.4 \end{Bmatrix}$$

<u>Reformulation</u>

$$\text{Min}\, z = y \quad \text{(A7.32.4)}$$

s. t. $w_i = (x_i - b_i)^2, \;\; i \in \{1,..,n\}$ (A7.32.5)

$$u = \sum_{i=1}^{n} w_i \quad \text{(A7.32.6)}$$

$$d = \sqrt{u} \quad \text{(A7.32.7)}$$

$$D \ge \sqrt{n}(x_i - b_i), \;\; i \in \{1,..,n\} \quad \text{(A7.32.8)}$$

$$D \ge \sqrt{n}(b_i - x_i), \;\; i \in \{1,..,n\} \quad \text{(A7.32.9)}$$

$$\mathrm{y}_1 = -\frac{1.0 + 0.2d}{D + 0.1} \quad \text{(A7.32.10)}$$

$$\mathrm{y}_2 = \cos(D\pi)\exp\left(-D/2\pi\right) \quad \text{(A7.32.11)}$$

$$\mathrm{y} = \mathrm{y}_1\mathrm{y}_2 \quad \text{(A7.32.12)}$$

We used one-dimensional grids to linearize Equations (A7.32.5), (A7.32.7), and (A7.32.11); and two-dimensional grids to linearize Equations (A7.32.10) and (A7.32.12).

33) Paviani

$$\text{Min}\, z = \sum_{i=1}^{10}\left(\left(\ln(x_i - 2)\right)^2 + \left(\ln(10 - x_i)\right)^2 \right)^2 - \left(\prod_{i=1}^{10} x_i \right)^{0.2} \tag{A7.33.1}$$

$$s.t.\ 2 \le x_i \le 10,\ i \in \{1,..,10\}$$

<u>Reformulation</u>

$$\text{Min}\, z = \sum_{i=1}^{10}\left(\left(\ln(x_i - 2)\right)^2 + \left(\ln(10 - x_i)\right)^2 \right)^2 - \exp\left(0.2 \log\left(\prod_{i=1}^{10} x_i \right)\right) \tag{A7.33.2}$$

Or $$\text{Min}\, z = \sum_{i=1}^{10}\left(\left(\ln(x_i - 2)\right)^2 + \left(\ln(10 - x_i)\right)^2 \right)^2 - \exp\left(0.2 \sum_{i=1}^{10} \log(x_i) \right) \tag{A7.33.3}$$

Or $$\text{Min}\, z = \sum_{i=1}^{n} y_i - v \tag{A7.33.4}$$

$$y_i = \left(\ln(x_i - 2)\right)^2 + \left(\ln(10 - x_i)\right)^2,\ \ i \in \{1,..,n\} \tag{A7.33.5}$$

$$w_i = \log(x_i),\ \ i \in \{1,..,n\} \tag{A7.33.6}$$

$$w = \sum_{i=1}^{n} w_i \tag{A7.33.7}$$

$$v = \exp(0.2w) \tag{A7.33.8}$$

We used one-dimensional grids to linearize Equations (A7.33.5), (A7.33.6), and (A7.33.8).

34) Periodic Problem

$$\text{Min}\, z = 1 + \sin^2(x_1) + \sin^2(x_2) - 0.1\exp\left(-x_1^2 - x_2^2\right) \tag{A7.34.1}$$

$$s.t.\ -10 \le x_i \le 10,\ \ i \in \{1,2\}$$

We used a two-dimensional grid to linearize Equation (A7.34.1).

35) Powell’s Quadratic Problem

$$\text{Min}\, z = (x_1 + 10x_1)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \quad \text{(A7.35.1)}$$

$s.t.\ -10 \le x_i \le 10,\ \ i \in \{1, ..., 4\}$

<u>Reformulation</u>

$$\text{Min}\, z = y_1 + y_2 + y_3 + y_4 \quad \text{(A7.35.2)}$$

$$y_1 = (x_1 + 10x_1)^2 \quad \text{(A7.35.3)}$$

$$y_2 = 5(x_3 - x_4)^2 \quad \text{(A7.35.4)}$$

$$y_3 = (x_2 - 2x_3)^4 \quad \text{(A7.35.5)}$$

$$y_4 = 10(x_1 - x_4)^4 \quad \text{(A7.35.6)}$$

We used a one dimensional grid to linearize Equation (A7.35.3) and two-dimensional grids to linearize Equations (A7.35.4–A7.35.6).

36) Price Transistor

We used two six-dimensional grids and one four-dimensional grid to linearize the nonlinear function in this problem.

$$\text{Min}\, z = \gamma^2 + \sum_{k=1}^{4} \left(\alpha_k^2 + \beta_k^2\right) \quad \text{(A7.36.1)}$$

$$\alpha_k = (1 - x_1x_2)x_3\{\exp[x_5(g_{1k} - g_{3k} * 0.001 * x_7 - g_{5k} * 0.001 * x_8)] - 1\} - g_{5k} + g_{4k}x_2$$

$$\text{for } k = 1,...,4 \quad \text{(A7.36.2)}$$

$$\beta_k = (1 - x_1x_2)x_4\{\exp[x_6(g_{1k} - g_{2k} - g_{3k} * 0.001 * x_7 + g_{4k} * 0.001 * x_9)] - 1\} - g_{5k}x_1 + g_{4k}$$

$$\text{for } k = 1,...,4 \quad \text{(A7.36.3)}$$

$$\gamma = x_1x_3 - x_2x_4 \quad \text{(A7.36.4)}$$

i	g_{ik}			
	$k=1$	2	3	4
1	0.485	0.752	0.869	0.982
2	0.369	1.254	0.703	1.455
3	5.2095	10.0677	22.9274	20.2153
4	23.3037	101.779	111.461	191.267
5	28.5132	111.8467	134.3884	211.4823

Table A7.7: Data for Price Transistor

Reformulation

$$\text{Min}\, z = t + \sum_{k=1}^{4} (p_k + q_k) \tag{A7.36.5}$$

$$u = (1 - x_1 x_2) x_3 \tag{A7.36.6}$$

$$v = (1 - x_1 x_2) x_4 \tag{A7.36.7}$$

$$r_k = g_{1k} - g_{3k} * 0.001 * x_7 - g_{5k} * 0.001 * x_8 \text{ for } k = 1,\ldots,4 \tag{A7.36.8}$$

$$s_k = g_{1k} - g_{2k} - g_{3k} * 0.001 * x_7 + g_{4k} * 0.001 * x_9 \text{ for } k = 1,\ldots,4 \tag{A7.36.9}$$

$$p_k = \left(u\{\exp[x_5 r_k] - 1\} - g_{5k} + g_{4k} x_2\right)^2 \text{ for } k = 1,\ldots,4 \tag{A7.36.10}$$

$$q_k = \left(v\{\exp[x_6 s_k] - 1\} - g_{5k} x_1 + g_{4k}\right)^2 \text{ for } k = 1,\ldots,4 \tag{A7.36.11}$$

$$t = (x_1 x_3 - x_2 x_4)^2 \tag{A7.36.12}$$

We used three-dimensional grids for Equations (A36.6) and (A36.7), and four-dimensional grids for linearizing Equations (A36.10–A36.12).

37) Rastrigin Problem

$$\text{Min}\, z = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right) \tag{A7.37.1}$$

$$s.t.\ -5.12 \le x_i \le 5.12,\ i \in \{1,\ldots,n\}$$

<u>Reformulation</u>

$$\text{Min}\, z = 10n + \sum_{i=1}^{n} y_i \tag{A7.37.2}$$

$$y_i = x_i^2 - 10\cos(2\pi x_i), \quad i \in \{1,..,n\} \tag{A7.37.3}$$

We used a one-dimensional grid to linearize Equation (A7.37.3).

38) Rosenbrock Problem

$$\text{Min}\, z = \sum_{i=1}^{n-1} \left(\left(x_{i+1} - x_i^2 \right)^2 + \left(x_i - 1 \right)^2 \right) \tag{A7.38.1}$$

$$s.t. \; -30 \le x_i \le 30, \; i \in \{1,...,n\}$$

<u>Reformulation</u>

$$\text{Min}\, z = \sum_{i=1}^{n-1} w_i \tag{A7.38.2}$$

$$w_i = \left(x_{i+1} - x_i^2 \right)^2 + \left(x_i - 1 \right)^2, \quad i \in \{1,..,n-1\} \tag{A7.38.3}$$

We used two-dimensional grids to linearize Equations (A7.38.3).

39) Salomon Problem

$$\text{Min}\, z = 1 - \cos(2\pi \| x \|) + 0.1 \| x \| \tag{A7.39.1}$$

$$\text{where} \| x \| = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{A7.39.2}$$

$$s.t. \; -100 \le x_i \le 100, \; i \in \{1,...,n\}$$

<u>Reformulation</u>

$$\text{Min}\, z = 1 - u \tag{A7.39.3}$$

$$y_i = x_i^2, \quad i \in \{1,..,n\} \tag{A7.39.4}$$

$$y = \sum_{i=1}^{n} y_i \tag{A7.39.5}$$

$$u = \cos\left(2\pi\sqrt{y}\right) - 0.1\sqrt{y} \tag{A7.39.6}$$

We used one-dimensional grids to linearize Equations (A7.39.4) and (A7.39.6).

40) Schaffer 1

$$\text{Min}\, z = 0.5 + \frac{\left(\sin\left(\sqrt{x_1^2 + x_2^2}\right)\right)^2 - 0.5}{\left(1 + 0.001\left(x_1^2 + x_2^2\right)\right)^2} \tag{A7.40.1}$$

$$s.t.\ -100 \le x_i \le 100,\ \ i \in \{1, 2\}$$

We used a two-dimensional grid to linearize Equations (A7.40.1).

41) Schaffer 2

$$\text{Min}\, z = \left(x_1^2 + x_2^2\right)^{0.25}\left(\sin^2\left(50\left(x_1^2 + x_2^2\right)^{0.1}\right) + 1\right) \tag{A7.41.1}$$

$$s.t.\ -100 \le x_i \le 100,\ \ i \in \{1, 2\}$$

We used a two-dimensional grid to linearize Equations (A7.41.1).

42) Shubert

$$\text{Min}\, z = \prod_{i=1}^{n}\left(\sum_{j=1}^{5}\left(j\cos\left((j+1)x_i + j\right)\right)\right) \tag{A7.42.1}$$

$$s.t.\ -10 \le x_i \le 10,\ \ i \in \{1, \ldots, n\}$$

We used a two-dimensional grid to linearize Equations (A7.42.1) for $n = 2$.

43) Schwefel Problem

$$\text{Min}\, z = -\sum_{j=1}^{n} \left(x_i \sin\left(\sqrt{|x_i|}\right)\right) \tag{A7.43.1}$$

$$s.t.\ -500 \le x_i \le 500,\ \ i \in \{1,\dots,n\}$$

<u>Reformulation</u>

$$\text{Min}\, z = \sum_{i=1}^{n} y_i \tag{A7.43.2}$$

$$y_i = x_i \sin\left(\sqrt{|x_i|}\right)$$

We used one-dimensional grids to linearize Equations (A7.43.2).

44) Shekel 5 Problem

$$\text{Min}\, z = -\sum_{i=1}^{5} \left(\frac{1}{c_i + \sum_{j=1}^{4} \left(x_j - a_{ij}\right)^2} \right) \qquad s.t.\ 0 \le x_i \le 10,\ \ j \in \{1,\dots,4\} \tag{A7.44.1}$$

Problem	*i*	a_{ij}				c_i
		j=1	*j*=2	*j*=3	*j*=4	
S5	1	4	4	4	4	0.1
	2	1	1	1	1	0.2
	3	8	8	8	8	0.2
	4	6	6	6	6	0.4
	5	3	7	3	7	0.4
S7	6	2	9	2	9	0.6
	7	5	5	3	3	0.3
	8	8	1	8	1	0.7
S10	9	6	2	6	2	0.5
	10	7	3.6	7	3.6	0.5

Table A7.8: Data for Shekel Problems

Reformulation

$$\text{Min}\, z = -\sum_{i=1}^{5} u_i \tag{A7.44.2}$$

$$u_i = \frac{1}{w_i},\ i \in \{1,...,5\} \tag{A7.44.3}$$

$$w_i = c_i + \sum_{j=1}^{4} y_j + (a_{ij})^2 - 2a_{ij}x_j\ ,\ i = 1,...,5 \tag{A7.44.4}$$

$$y_j = (x_j)^2,\ \ j \in \{1,...,4\} \tag{A7.44.5}$$

We used one-dimensional grids to linearize Equations (A7.44.3) and (A7.44.5).

45) Shekel 7 Problem

$$\text{Min}\, z = -\sum_{i=1}^{7} \left(\frac{1}{c_i + \sum_{j=1}^{4} \left(x_j - a_{ij}\right)^2} \right) \tag{A7.45.1}$$

$$s.t.\ 0 \le x_i \le 10,\ \ j \in \{1,...,4\}$$

Reformulation

$$\text{Min}\, z = -\sum_{i=1}^{7} u_i \tag{A7.45.2}$$

$$u_i = \frac{1}{w_i},\ i \in \{1,...,7\} \tag{A7.45.3}$$

$$w_i = c_i + \sum_{j=1}^{4} y_j + (a_{ij})^2 - 2a_{ij}x_j\ ,\ i = 1,...,7 \tag{A7.45.4}$$

$$y_j = (x_j)^2,\ \ j \in \{1,...,4\} \tag{A7.45.5}$$

We used one-dimensional grids to linearize Equations (A7.45.3) and (A7.45.5).

46) Shekel 10 Problem

$$\text{Min}\, z = -\sum_{i=1}^{10}\left(\frac{1}{c_i + \sum_{j=1}^{4}\left(x_j - a_{ij}\right)^2}\right) \tag{A7.46.1}$$

$s.t.\ 0 \le x_i \le 10,\ \ j \in \{1,\ldots,4\}$

<u>Reformulation</u>

$$\text{Min}\, z = -\sum_{i=1}^{10} u_i \tag{A7.46.2}$$

$$u_i = \frac{1}{w_i},\ \ i \in \{1,\ldots,10\} \tag{A7.46.3}$$

$$w_i = c_i + \sum_{j=1}^{4} y_j + (a_{ij})^2 - 2a_{ij}x_j\,,\ i = 1,\ldots,10 \tag{A7.46.4}$$

$$y_j = (x_j)^2,\ \ j \in \{1,\ldots,4\} \tag{A7.46.5}$$

We used one-dimensional grids to linearize Equations (A7.45.3) and (A7.45.5).

47) Sinusoidal Problem

$$\text{Min}\, z = -\left[A\prod_{i=1}^{n}\sin\left(x_i - C\right) + \prod_{i=1}^{n}\sin\left(B\left(x_i - C\right)\right)\right] \tag{A7.47.1}$$

$s.t.\ 0 \le x_i \le 180,\ \ i \in \{1,\ldots,n\}$

Our tests were performed with A = 2.5, B = 5, C = 30, n= 20.

<u>Reformulation</u>

$$\text{Min}\, z = -\left[Ay + s\right] \tag{A7.47.2}$$

$$u_i = \sin\left(x_i - C\right),\ \ i \in \{1,\ldots,20\} \tag{A7.47.3}$$

$$v_j = u_{2j-1}u_{2j}\,,\ \ j \in \{1,\ldots,10\} \tag{A7.47.4}$$

$$w_k = v_{2k-1}v_{2k}\,,\ \ k \in \{1,\ldots,5\} \tag{A7.47.5}$$

$$y_1 = w_1 w_2 \tag{A7.47.6}$$

$$y_2 = w_3 w_4 \tag{A7.47.7}$$

$$y_3 = y_1 y_2 \tag{A7.47.8}$$

$$y = y_3 w_5 \tag{A7.47.9}$$

$$p_i = \sin\left(B\left(x_i - C\right)\right),\ \ i \in \{1,\ldots,n\} \tag{A7.47.10}$$

$$q_j = p_{2j-1} p_{2j},\ \ j \in \{1,\ldots,10\} \tag{A7.47.11}$$

$$r_k = q_{2k-1} q_{2k},\ \ k \in \{1,\ldots,5\} \tag{A7.47.12}$$

$$s_1 = r_1 r_2 \tag{A7.47.13}$$

$$s_2 = r_3 r_4 \tag{A7.47.14}$$

$$s_3 = s_1 s_2 \tag{A7.47.15}$$

$$s = s_3 r_5 \tag{A7.47.16}$$

We used one-dimensional grids to linearize Equations (A7.47.3) and (A7.47.10), and two dimensional grids to linearize (A7.47.4–A7.47.9) and (A7.47.12–A7.47.16).

48) Storn's Tchebychev Problem

$$\mathrm{Min}\, z = p_1 + p_2 + p_3 \tag{A7.48.1}$$

$$u = \sum_{i=1}^{n} \left(1.2\right)^{n-i} x_i \tag{A7.48.2}$$

$$p_1 = \begin{cases} \left(u - d\right)^2 & if\ u < d \\ 0 & if\ u \geq d \end{cases} \tag{A7.48.3}$$

$$v = \sum_{i=1}^{n} \left(-1.2\right)^{n-i} x_i \tag{A7.48.4}$$

$$p_2 = \begin{cases} \left(v - d\right)^2 & if\ v < d \\ 0 & if\ v \geq d \end{cases} \tag{A7.48.5}$$

$$w_j = \sum_{i=1}^{n} \left(\frac{2j}{m} - 1 \right)^{n-i} x_i \qquad \text{(A7.48.6)}$$

$$q_j = \begin{cases} (w_j - 1)^2 & if\ w_j > 1 \\ (w_j + 1)^2 & if\ w_j < -1 \\ 0 & if\ -1 \le w_j \le 1 \end{cases} \qquad \text{(A7.48.7)}$$

$$p_3 = \sum_{j=0}^{m} q_j \qquad \text{(A7.48.8)}$$

For $n = 9$: $x_i \in [-128, 128]^9$, $d = 72.661$, $m = 60$

For $n = 17$: $x_i \in [-32768, 32768]^{17}$, $d = 10558.145$, $m = 100$

<u>Reformulation</u>

For $n = 9$: $F := 256$

For $n = 17$: $F := 32678$

$$\text{Min}\, z' = p_1 + p_2 + p_3 \qquad \text{(A7.48.9)}$$

$$u = \sum_{i=1}^{n} (1.2)^{n-i} \left(\frac{x_i}{F} \right) \qquad \text{(A7.48.10)}$$

$$p_1' \ge \frac{d}{F} - u \qquad \text{(A7.48.11)}$$

$$p_1 = (p_1')^2 \qquad \text{(A7.48.12)}$$

$$v = \sum_{i=1}^{n} (-1.2)^{n-i} \left(\frac{x_i}{F} \right) \qquad \text{(A7.48.13)}$$

$$p_2' \ge \frac{d}{F} - v \qquad \text{(A7.48.14)}$$

$$p_2 = (p_2')^2 \qquad \text{(A7.48.15)}$$

$$w_j = \sum_{i=1}^{n} \left(\frac{2j}{m} - 1 \right)^{n-i} \left(\frac{x_i}{F} \right) \ \text{for } j = 0, \dots, m \qquad \text{(A7.48.16)}$$

$$r_j' \ge w_j - \frac{1}{F} \ \text{for } j = 0, \dots, m \qquad \text{(A7.48.17)}$$

$$r_j = (r_j')^2 \ \text{for } j = 0, \dots, m \qquad \text{(A7.48.18)}$$

$$s_j' \geq -w_j - \frac{1}{F} \quad \text{for } j = 0,\ldots,m \tag{A7.48.19}$$

$$s_j = (s_j')^2 \quad \text{for } j = 0,\ldots,m \tag{A7.48.20}$$

$$q_j = r_j + s_j \quad \text{for } j = 0,\ldots,m \tag{A7.48.21}$$

$$p_3 = \sum_{j=0}^{m} q_j \tag{A7.48.22}$$

49) Wood's function

$$\begin{aligned} \text{Min}\, z = {} & 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 + 90\left(x_4 - x_3^2\right)^2 + (1 - x_3)^2 \\ & + 10.1\left[(x_2 - 1)^2 + (x_4 - 1)^2\right] + 19.8\left[(x_2 - 1)(x_4 - 1)\right] \end{aligned} \tag{A7.49.1}$$

$$s.t.\ -10 \leq x_i \leq 10,\ \ i \in \{1,\ldots,4\}$$ ‘

<u>Reformulation</u>

$$\text{Min}\, z = y_1 + y_2 + y_3 \tag{A7.49.2}$$

$$\text{y}_1 = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 \tag{A7.49.3}$$

$$\text{y}_2 = 90\left(x_4 - x_3^2\right)^2 + (1 - x_3)^2 \tag{A7.49.4}$$

$$\text{y}_5 = 10.1\left[(x_2 - 1)^2 + (x_4 - 1)^2\right] + 19.8\left[(x_2 - 1)(x_4 - 1)\right] \tag{A7.49.5}$$

We used two-dimensional grids to linearize Equations (A7.49.3–A7.48.5).

Appendix A8: Determining the Best Non-uniform Grid for an *n*-dimensional Function

Given a *n*-dimensional function $f(x_1,...,x_n)$, $l_i \le x_i \le u_i$ and a set of break points along each dimension, partition the domain along each dimension into segments and the entire space into *n*-dimensional boxes such that the sum of the approximation errors over all the boxes is minimized. The approximation error arises when the function values at the points that lie on or inside a box are approximated by convex combination of the function values at the vertices of the box.

<u>Sets and parameters</u>

D	set of dimensions
d	index over D
n	number of dimensions
I^d	set of intervals along dimension d
S^d	set of segments along dimension d
i	index over I^d for all $d \in D$
s	index over S^d for all $d \in D$
α_{is}^d	parameter that is one if interval $i \in I^d$ belongs to segment $s \in S^d$ along dimension d, and is zero otherwise
B	set of *n*-dimensional boxes
b	index over B
c_b	approximation error (or cost) associated with box b
$s(b,d)$	segment corresponding to box b along dimension d
σ^d	number of segments required along dimension d

$B(s,d)$ set of boxes corresponding to segment s along dimension d

β total number of boxes required; $\beta = \prod_{d \in D} \sigma^d$

Π set of all possible interval combinations along all the dimensions ;

$\Pi = I^1 \times I^2 \times \ldots \times I^n$

π index over Π; each π can be conceptualized as a box in *n*-dimensional space such that each edge of this box corresponds to an interval. We call such a box a *pixel*.

$B(\pi)$ set of boxes that cover pixel π

Variables

X_s^d binary variable that is one if we select segment s along dimension d , and is zero otherwise

Y_b binary variable that is one if we select box b, and is zero otherwise

Objective

Our goal is to minimize the sum of the approximation errors over each selected box

$$\text{Min} \quad \sum_{b \in B} c_b Y_b \qquad \text{(A8.1)}$$

Constraints

1) Interval selection constraint

Each interval along each dimension must be covered by exactly one segment.

$$\sum_{s \in S^d} \alpha_{is}^d X_s^d = 1 \quad \text{for all } i \in I^d,\ d \in D \qquad \text{(A8.2)}$$

2) Segment selection constraint

For each dimension, the number of segments selected should be equal to desired number of segments along that dimension.

$$\sum_{s \in S^d} X_s^d = \sigma^d \quad \text{for all } d \in D \tag{A8.3}$$

3) Box selection constraint

For each box, if we select each of the segments that the box corresponds to along each dimension, then we must select that box.

$$Y_b \geq \sum_{d \in D} X_{s(b,d)}^d - n + 1 \quad \text{for all } b \in B \tag{A8.4}$$

4) Total number of boxes constraint

The total number of boxes selected must be equal to the required number of boxes.

$$\sum_{b \in B} Y_b = \beta \tag{A8.5}$$

5) Box forcing constraint

If we select a segment along dimension *d*, then we must select a specified number of boxes (equal to β / σ^d) along that segment.

$$\sum_{b \in B(s,d)} Y_b = \left(\frac{\beta}{\sigma^d} \right) X_s^d \quad \text{for all } s \in S^d,\ d \in D \tag{A8.6}$$

6) Pixel covering constraint

Each pixel in *n*-dimensional space must be covered by exactly one box.

$$\sum_{b \in B(\pi)} Y_b = 1 \quad \text{for all } \pi \in \Pi \tag{A8.7}$$

7) Binary Variable constraint

The segment selection and box selection constraints are binary.

$$X_s^d \in \{0,1\} \quad \text{for all } d \in D,\ s \in S^d \tag{A8.8}$$

$$Y_b \in \{0,1\} \quad \text{for all } b \in B \tag{A8.9}$$

Note: If we are using the maximum error metric, then we define:

- a parameters $\hat{c}_b$ which indicates the approximation error associated with box b based on the maximum error metric, and
- a continuous variable W that indicates the maximum error over all the selected boxes.

We need a new objective as given by Equation (A8.10) and a new constraint as given by Equation (A8.11). The model is comprised of Equations (A8.2 –A8.11).

$$\text{Min } W \tag{A8.10}$$

$$\text{W} \geq \hat{c}_b Y_b \quad \text{for all } b \in B \tag{A8.11}$$

<u>Strengthening the aggregate box selection inequality</u>

We can strengthen the box selection constraint by aggregating it over multiple boxes that have the property that they cannot all be present in a feasible integer solution. For each dimension $d \in D$, let us define Ω^d as the set of all possible combinations of segments along that dimension such that in each combination, the segments cover a common interval. This implies that in a feasible integer solution, no more than one segment can be selected from a segment combination.

Let $\Pi = \Omega^1 \times \Omega^2 \times ... \times \Omega^{|D|}$ be the product of the segment combinations along each dimension and let π be an index over Π. Each combination product π corresponds to

(a) a set of segment-combinations along each dimension such each segment combination shares a common interval, and

(b) a set of boxes which share a common space.

Further, let $S^d(\pi)$ be the set of segments along dimension d that are present in the combination product π, and $B(\pi)$ be the set of boxes formed by the set of segments in the combination product π. Then, the *aggregate box selection inequality* is given by Equation (A8.9).

$$\sum_{b\in B(\pi)} Y_b \geq \sum_{d\in D} \sum_{s\in S^d(\pi)} X_s^d - n + 1 \quad \text{for all } \pi \in \Pi \tag{A8.12}$$

This inequality is valid because (a) on the right hand side, for each dimension, no more than one segment can be selected, since the segments cover a common interval, and (b) on the left hand side, no more than one box can be chosen depending upon the segments chosen along each dimension. We now provide an LP solution that does not satisfy the *aggregate box selection inequality*.

<u>LP solution violating the box selection inequality</u>

Consider a scenario in which we have a two-dimensional grid with two intervals along each dimension, and we wish to create a grid with two segments along each dimension. In this case, the only solution is to choose each interval as a segment. Let us now see how the mixed integer program defined by Equations (A8.1) – (A8.6) will solve this problem. To do so, we first need to define some sets and parameters.

<u>Notation</u>

I^1, I^2	set of intervals along Dimension 1 and Dimension 2 respectively
p, q	indices over I^1
r, s	indices over I^2
S^1	set of segments along Dimension 1; a segment covering intervals *p* through *q* is denoted by the ordered pair (*p*, *q*)
S^2	set of segments along Dimension 2; a segment covering intervals *r* through *s* is denoted by the ordered pair (*r*, *s*)

B	set of boxes; a box formed by segments (*p*, *q*) along Dimension 1 and (*r*, *s*) along Dimension 2 is denoted by the 4-tuple (*p*, *q*, *r*, *s*)
X^1_{pq}	binary variable that is 1 if we select segment (*p*, *q*) along Dimension 1, and zero otherwise
X^2_{rs}	binary variable that is 1 if we select segment (*r*, *s*) along Dimension 2, and zero otherwise
Y_{pqrs}	binary variable that is 1 if we select box (*p*, *q*, *r*, *s*), and zero otherwise

In the current case, where we have a 2x2 grid, the various sets can be listed as follows.

$$I^1 = \{1,2\},\ I^2 = \{1,2\}$$

$$S^1 = \{(1,1),(1,2),(2,2)\},\ S^2 = \{(1,1),(1,2),(2,2)\}$$

Consider the following *LP* solution.

$$X^1_{pq} = \frac{1}{2} \quad \text{for all} \quad (p,q) \in \{(1,1),(1,2),(2,2)\} \qquad \text{(A8.13a)}$$

$$X^2_{rs} = \frac{1}{2} \quad \text{for all} \quad (r,s) \in \{(1,1),(1,2),(2,2)\} \qquad \text{(A8.13b)}$$

$$Y_{pqrs} = 0 \quad \text{for all} \quad (p,q) \in \{(1,1),(1,2),(2,2)\}, \quad (r,s) \in \{(1,1),(1,2),(2,2)\}$$

(A8.13c)

This solution satisfies the box selection constraint but since it does not assign a strictly positive value to any of the box selection variables, it can achieve an objective value of zero. However, if we apply the aggregate box inequality for the overlapping segments sets $\Omega^1 = \{(1,1),(1,2)\}$ and $\Omega^2 = \{(1,1),(1,2)\}$, we get the Equation (A8.11).

$$Y_{1111}+Y_{1112}+Y_{1211}+Y_{1112} \geq X^1_{11}+X^1_{12}+X^2_{11}+X^2_{12}-1 \qquad \text{(A8.14)}$$

The left hand side of this equation has a value of 1 and therefore it forces one or more of the box selection variables (on the left hand side) to take a strictly positive value. Thus, the *LP* solution given by Equation (A8.10) will be removed by the aggregate box inequality.

Appendix A9: Models and Reformulations for Gasnet

The *gasnet* model involves designing a gas pipeline system that can transport a fixed amount of gas from point *A* to points *B* and *C*. The pressure, temperature and composition of the gas at points *A*, *B*, and *C* are known. The configuration of the pipeline system is shown in Figure 1. There are a total of twelve pipeline segments and ten compressors. Each segment has five associated variables: the flow rate (*Q*), the inlet pressure (p_d) which is also the discharge pressure from the upstream compressor, the outlet pressure (p_s) which is also the suction pressure for the downstream compressor, the pipe diameter (*D*), and the length of the pipeline segment (*L*). The objective is to design a pipe at the lowest cost.

<u>Model-I</u>

Sets

I	set of segments of pipes,	i, j	indices over I
C	set of compressors	c	index over C
B	set of branches	b	index over B

$I_u(c)$	set of pipe segments that are immediately upstream of compressor *c*
$I_d(c)$	set of pipe segments that are immediately downstream of compressor *c*
$I_u(i)$	set of pipe segments that are immediately upstream of segment *i*
$I_d(i)$	set of pipe segments that are immediately downstream of segment *i*
$c(i)$	compressor downstream of segment *i*
$I_{nonterm}$	set of non-terminal pipe segments
$I(b)$	set of pipe segments that belong to branch *b*

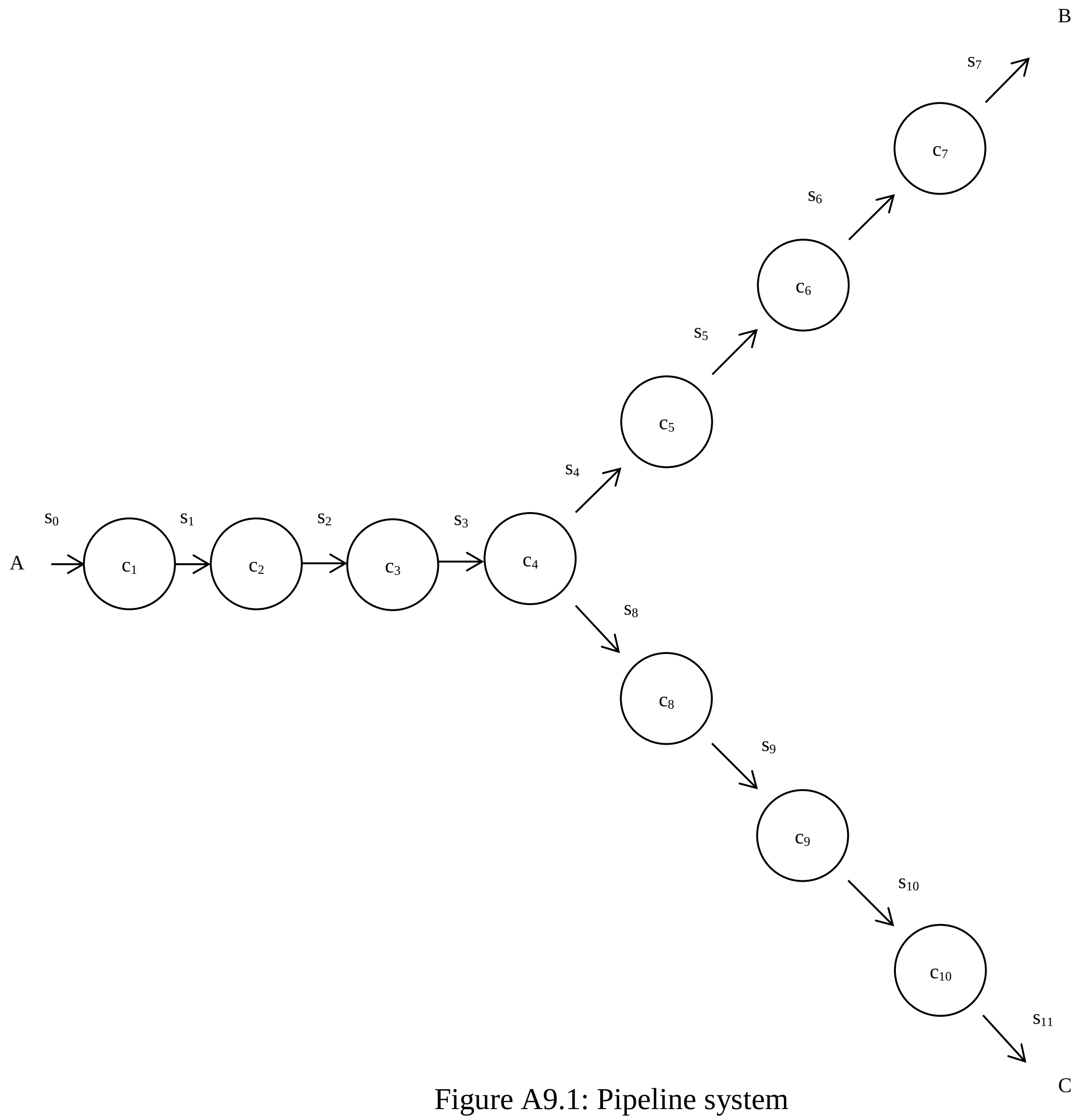


Figure A9.1: Pipeline system

Parameters

k	ratio of specific heat at constant pressure to the specific heat at constant volume at suction conditions, assumed to be 1.26

θ	compressibility factor of gas at suction conditions, assumed to be 0.88
t	suction temperature in ^{o}R, assumed to be 520
σ_{fix}	fixed cost for a compressor
σ_{var}	variable cost for a compressor, expressed in dollars per unit work done by the compressor
σ_{op}	operating cost for a compressor, expressed in dollars per unit work done by the compressor
π	fixed cost for pipe, expressed in dollars per unit length per unit diameter of the pipe
λ_b	length of branch *b*
$\rho_c^{\max}$	maximum compression ratio allowed for compressor *c*

Variables

L_i	length of pipe segment *i*
P_{d_i}	discharge pressure for pipe segment *i*
P_{s_i}	suction pressure for pipe segment *i*
D_i	diameter of pipe segment *i*
Q_i	flow in pipe segment *i*
ρ_c	compression ratio for compressor *c*
W_c	work done by compressor *c*
Y_c	binary variable that takes value 1 if compressor *c* is selected, and takes value zero otherwise

Objective

Our objective is to minimize the sum of the fixed, variable and operating costs of the compressors and the fixed pipe costs.

$$\text{Min} \sum_{i\in I} \pi\ L_i D_i + \sum_{c\in C} \sigma_{fix}\ Y_c + \sum_{c\in C} (\sigma_{\text{var}} + \sigma_{op})\ Y_c \qquad \text{(A9.1)}$$

Constraints

1) Pressure drop in pipe segments

For each pipe, the discharge pressure must be greater than or equal to the suction pressure.

$$P_{d_i} - P_{s_i} \geq 0 \quad \text{for all } i \in I \qquad \text{(A9.2)}$$

2) Length of branches

The total length of the segments in a particular branch must be equal to the specified branch length.

$$\sum_{i\in I(b)} L_i = \lambda_b \quad \text{for all} \quad b \in B \qquad \text{(A9.3)}$$

3) Flow Equation

The flow equation links the flow rate, pipe length, pipe diameter, discharge pressure and suction pressure for the pipe segment.

$$Q_i^{\ 2} = \frac{1}{L_i}(0.871)^2 (D_i)^{16/3} \left(P_{d_i}^{\ 2} - P_{s_i}^{\ 2}\right) \ \text{for all } i \in I \qquad \text{(A9.4)}$$

or

$$L_i = \frac{1}{Q_i^{\ 2}}(0.871)^2 (D_i)^{16/3} \left(P_{d_i}^{\ 2} - P_{s_i}^{\ 2}\right) \ \text{for all } i \in I \qquad \text{(A9.5)}$$

4) Compressor work definition

The compressor work definition expresses the work done by a compressor in terms of the suction temperature (*t*), the ratio of specific heat at constant pressure to the specific heat at constant volume (*k*), the compressibility factor of gas at suction conditions (θ), the compression ratio for compressor (ρ), and the flow rate through the compressor.

$$W_c = 0.08531\ t\ Q_i \frac{k}{k-1} \left(\rho^{\theta(k-1)/k} - 1\right) \quad \text{for all } c \in C,\ i \in I_d(c) \qquad \text{(A9.6)}$$

5) Maximum compression

For each compressor, the compression ratio must be between 1and the maximum compression ratio for that compressor.

$$\rho_c \le 1 + \left(\rho_c^{\max} - 1\right) Y_c \quad \text{for all } c \in C \qquad \text{(A9.7)}$$

6) Compression definition

For each compressor, the compression ratio is the ratio of the discharge pressure into the downstream pipe segment and the suction pressure in the upstream pipe.

$$\rho_c = \frac{P_{d_i}}{P_{s_j}} \quad \text{for all } c \in C,\ i \in I_u(c),\ j \in I_d(c) \qquad \text{(A9.8)}$$

7) Flow balance

The flow rate out of a compressor is 99.5% of the flow rate into the compressor.

$$Q_i - \sum_{j \in I_d(i)} Q_j = 0.005\ Q_i\, Y_{c(i)} \quad \text{for all } i \in I_{nonterm} \qquad \text{(A9.9)}$$

<u>**Reformulation**</u>

We present four different models of the gasnet problem.

<u>Model 0</u>

1) We use a two dimensional grid for linearizing the first term in Equation (A9.1).
2) We use a four dimensional grid to linearize Equation (A9.5).
3) We use a two dimensional grid to linearize Equation (A9.6), Equation (A9.7) and Equation (A9.8).

Model 1

1) We use a two dimensional grid for linearizing the first term in Equation (A9.1).

2) We use a four dimensional grid to linearize Equation (A9.5).

3) We use a two dimensional grid to linearize Equation (A9.6) and Equation (A9.8).

4) Let $\hat{Q}_i$ be the upper bound on Q_i. Then, instead of creating a two dimensional grid for Equation (A3.9), we use the transformation given by Equations (A9.10–A9.13) which ensures that

   a. when $Y_{c(i)}$ is equal to one, then $\sum_{j \in I_d(i)} Q_j = 0.995\, Q_i$, and

   b. when $Y_{c(i)}$ is equal to zero, then $\sum_{j \in I_d(i)} Q_j = Q_i$.

$$\sum_{j \in I_d(i)} Q_j - 0.995\, Q_i \le 0.005\, \hat{Q}_i (1 - Y_{c(i)}) \quad \text{for all } i \in I_{nonterm} \qquad \text{(A9.10)}$$

$$0.995\, Q_i \le \sum_{j \in I_d(i)} Q_j \le Q_i \quad \text{for all } i \in I_{nonterm} \qquad \text{(A9.11)}$$

$$Q_i - \sum_{j \in I_d(i)} Q_j \le 0.005 \hat{Q}_i Y_{c(i)} \quad \text{for all } i \in I_{nonterm} \qquad \text{(A9.12)}$$

When $Y_{c(i)}$ is equal to one, Equation (A9.10) becomes Equation (A9.13), which together with Equation (A9.11) leads to Equation (A9.14), which is condition (a) above.

$$\sum_{j \in I_d(i)} Q_j \le 0.995\, Q_i \quad \text{for all } i \in I_{nonterm} \qquad \text{(A9.13)}$$

$$\sum_{j \in I_d(i)} Q_j = 0.995\, Q_i \quad \text{for all } i \in I_{nonterm} \qquad \text{(A9.14)}$$

When $Y_{c(i)}$ is equal to zero, Equation (A9.12) can be replaced by Equation (A9.15), which together with Equation (A9.11) leads to Equation (A9.16), which is condition (b) above.

$$Q_i \le \sum_{j \in I_d(i)} Q_j \quad \text{for all } i \in I_{nonterm} \qquad \text{(A9.15)}$$

$$\sum_{j \in I_d(i)} Q_j = Q_i \quad \text{for all } i \in I_{nonterm} \qquad \text{(A9.16)}$$

Model-II

Model-II is the same as Model I except that we make the following changes.

a) Replace the P_{d_i} variables by a new set of variables $\overline{P}_{d_i}$ such that $\overline{P}_{d_i} = -P_{d_i}$ for all $i \in I$.

b) Replace Equation (A9.2) with Equation (A9.17) which contains $\overline{P}_{d_i}$ variables.

$$L_i = \frac{1}{Q_i^2}(0.871)^2 (D_i)^{16/3}\left(\overline{P}_{d_i}^{\ 2} - P_{s_i}^{\ 2}\right) \text{ for all } i \in I \qquad \text{(A9.17)}$$

c) Replace Equation (A9.5) Equation (A9.18) which contains $\overline{P}_{d_i}$ variables.

$$-\overline{P}_{d_i} - P_{s_i} \geq 0 \quad \text{for all } i \in I \qquad \text{(A9.18)}$$

d) Replace Equation (A9.8) by Equation (A9.19) which contains the $\overline{P}_{d_i}$ variables.

$$\rho_c = -\frac{\overline{P}_{d_i}}{P_{s_j}} \text{ for all } c \in C,\ i \in I_u(c),\ j \in I_d(c) \qquad \text{(A9.19)}$$

Model-II is comprised of Equations (A9.1), Equations (A9.3–A9.4), Equations (A9.6–A9.7), Equation (A9.9), and Equations (A9.17–A9.19).

We reformulate Model-II in the same manner as we reformulate Model-I. Further, we use two-dimensional grids to linearize Equations (A9.19).

Model-III

Model-III is the same as Model I except that we make the following changes.

a) Replace the P_{s_i} variables by a new set of variables $\overline{P}_{s_i}$ such that $\overline{P}_{s_i} = -P_{s_i}$ for all $i \in I$.

b) Replace Equation (A9.2) with Equation (A9.20) which contains the $\overline{P}_{s_i}$ variables.

$$L_i = \frac{1}{Q_i^2}(0.871)^2 (D_i)^{16/3}\left(P_{d_i}^{\ 2} - \overline{P}_{s_i}^{\ 2}\right) \text{ for all } i \in I \qquad \text{(A9.20)}$$

c) Replace Equation (A9.5) with Equation (A9.21) which contains the $\overline{P}_{s_i}$ variables.

$$P_{d_i} + \overline{P}_{s_i} \geq 0 \quad \text{for all } i \in I \qquad \text{(A9.21)}$$

d) Replace Equation (A9.8) with Equation (A9.22) which contains the $\overline{P}_{s_i}$ variables.

$$\rho_c = -\frac{P_{d_i}}{\overline{P}_{s_j}} \text{ for all } c \in C,\ i \in I_u(c),\ j \in I_d(c) \qquad \text{(A9.22)}$$

Model-III is comprised of Equations (A9.1), Equations (A9.3–A9.4), Equations (A9.6–A9.7), Equation (A9.9), and Equations (A9.20–A9.22).

We reformulate Model-III in the same manner as we reformulated Model-I. Further, we use two-dimensional grids to linearize Equations (A9.22).

Appendix A10: Model and Reformulation for Reactor Network Design

The *chemical reactor network design* model involves determining the types, sizes, and interconnections of reactors which optimize a desired performance objective. In the model considered here, the reactors are of the type Continuous Stirred Tank Reactors (CSTR).

Model

Sets

I	set of components,	i	indices over I
J	set of reactions	j	index over J
L	set of *CSTR* units	l	index over L
R	set of feeds	r	index over R
P	set of products	p	index over P

Parameters

v_{ij}	coefficient of reactant $i \in I$ in reaction $j \in J$
$\hat{k}_j$	rate constant, defined for $j \in J$
E_j	activation energy, defined for $j \in J$
R	gas constant
F_r^a	flow rate of feed streams, defined for $r \in R$
c_{ri}^a	flow rate of species in feed streams, defined for $r \in R,\ i \in I$
$i(j)$	reactant for reaction $j \in J$

Variables

F_{rl}^{ad}	flow rate from feed splitters to CSTR mixers, defined for $r \in R,\ l \in L$
F_l^d	flow rate into CSTR units, defined for $l \in L$

F_l^g	flow rate out of CSTR units, defined for $l \in L$
$F_{l,l'}^{gd}$	flow rate from CSTR outlets to CSTR inlets, defined for $l \in L,\ l' \in L$
F_{lp}^{gh}	flow rate from CSTR outlets to product mixers, defined for $l \in L,\ p \in P$
F_p^h	flow rate of product streams, defined for $p \in P$
c_{li}^d	concentration of species in the CSTR inlet streams, defined for $l \in L,\ i \in I$
c_{li}^g	concentration of species in the CSTR outlet streams, defined for $l \in L,\ i \in I$
c_{pi}^h	concentration of species in the product streams, defined for $p \in P,\ i \in I$
T_l^m	temperatures in the CSTR units, defined for $l \in L$
V_l^m	volumes of the CSTR units, defined for $l \in L$
r_{lj}^m	rate of reaction *j* in reactor *l*, defined for $l \in L,\ j \in J$

Objective

The current problem specifcations are as follows.

Set of components $I = \{A, B, C, D\}$

No. of reactions: 3

$$A \xrightarrow{\hat{k}_1} B \xrightarrow{\hat{k}_2} C$$

$$2A \xrightarrow{\hat{k}_3} D$$

No. of CSTR units: 5

No. of feeds: 1

No. of products: 1

Our objective is to maximize the yield of *B*.

$$\mathrm{Max}\, c_{1B}^h \qquad \text{(A10.1)}$$

Constraints

1) Flow conservation at the feed splitter

$$F_r^a = \sum_{l \in L} F_{rl}^{ad} \quad \text{for all } r \in R \tag{A10.2}$$

2) CSTR inlet mixer total balance

$$F_l^d = \sum_{r \in R} F_{rl}^{ad} + \sum_{l' \in L} F_{l'l}^{gd} \quad \text{for all } l \in L \tag{A10.3}$$

3) CSTR inlet mixer component balance

$$c_{li}^d F_l^d = \sum_{r \in R} c_{ri}^a F_{rl}^{ad} + \sum_{l' \in L} c_{l'i}^g F_{l'l}^{gd} \quad \text{for all } i \in I,\ l \in L \tag{A10.4}$$

4) CSTR total balance

$$F_l^g = F_l^d \quad \text{for all } l \in L \tag{A10.5}$$

5) CSTR component balance

$$c_{li}^g F_l^g = c_{li}^d F_l^d + V_l^m \sum_{j \in J} v_{ij} r_{lj}^m \quad \text{for all } i \in I,\ l \in L \tag{A10.6}$$

6) CSTR reaction rates

$$r_{lj}^m = \hat{k}_j \exp\left(-\frac{E_j}{RT_l^m}\right)\left(c_{li(j)}^g\right)^{v_{ij}} \text{ for all } j \in J,\ l \in L \tag{A10.7}$$

7) CSTR outlet splitter

$$F_l^g = \sum_{l' \in L} F_{ll'}^{gd} + \sum_{p \in P} F_{lp}^{gh} \text{ for all } l \in L \tag{A10.8}$$

8) Product mixer total balance

$$F_p^h = \sum_{l \in L} F_{lp}^{gh} \text{ for all } p \in P \tag{A10.9}$$

9) Product mixer component balance

$$c_{pi}^h F_p^h = \sum_{l \in L} c_{li}^g F_{lp}^{gh} \text{ for all } i \in I,\ p \in P \tag{A10.10}$$

Reformulation

1) We use two dimensional grids for linearizing the product terms in Equation (A10.4), Equation (A10.6), and Equation (A10.10), and the exponential terms in Equation (A10.7).

Appendix A11: LP Equivalence of Segment-wise and Cumulative adjacency

Given an NLP and its associated PLA reformulation, let [P] be the MIP model for the PLA problem. Formulation [P] can use either segment-wise adjacency or cumulative adjacency constraints for the combined partition of every variable, but we assume that it does not contain any additional constraints on the segment selection (z) variables such as valid inequalities relating z-values across grids. We consider the relationship between two relaxations of problem [P]: (i) the *LP relaxation* obtained by relaxing the integrality restrictions on the z-variables, and (ii) the *Adjacency relaxation* obtained by omitting the adjacency conditions and z-variables from [P].

Let V^{LP} and V^{Adj} denote the optimal values of these two relaxations. For any variable x of the original NLP problem, suppose the combined partition contains (n + 1) breakpoints (and associated) marginal weight variables, indexed from $i = 0, 1, \ldots, n$; the corresponding segments are indexed from $i = 1, 2, \ldots, n$ such that segment i extends from the (i – 1)st breakpoint to the ith breakpoint. For variable x, let $(\bar{\lambda}^x, \bar{z}^x)$ and $\hat{\lambda}^x$ respectively denote the vectors of marginal weights and segment selection values in these optimal solutions to the LP and Adjacency relaxations. (Based on the following result, we can show that the two relaxations have optimal solutions with the same optimal values for the vertex weight variables, i.e., α-variables. So, we do not consider these variables in our discussions.)

Proposition: $V^{LP} = V^{Adj}$, and, for every variable x, we can construct an optimal solution $\hat{\lambda}^x$ to the Adjacency relaxation from the solution $(\bar{\lambda}^x, \bar{z}^x)$ to the LP relaxation, and vice versa.

Proof: For any *x*, given the solution $(\bar{\lambda}^x, \bar{z}^x)$ to the LP relaxation, the solution $\lambda^x = \bar{\lambda}^x$ is feasible for the Adjacency relaxation. Hence, $V^{LP} \geq V^{Adj}$.

Given the optimal solution $\hat{\lambda}^x$ to the Adjacency relaxation, consider the solution $\lambda^x = \hat{\lambda}^x$ and $z_1^x = \hat{\lambda}_0^x + \frac{1}{2}\hat{\lambda}_1^x$, $z_i^x = \frac{1}{2}\hat{\lambda}_{i-1}^x + \frac{1}{2}\hat{\lambda}_i^x$ for *i* = 2, 3, …., *n* – 1, and $z_n^x = \frac{1}{2}\hat{\lambda}_{n-1}^x + \hat{\lambda}_n^x$.

This solution (λ^x, z^x) is feasible for the LP relaxation, i.e., it satisfies the adjacency conditions (either cumulative or segment-wise) and the requirement that $\sum_{i=1}^{n} z_i^x = 1$. Hence, $V^{LP} \leq V^{Adj}$, implying that $V^{LP} = V^{Adj}$. The solution transformations above show how we can obtain an optimal solution for one relaxation from the optimal solution to the other.

Appendix A12: Notation for MIP Models

In this appendix, we discuss the notation for the individual grid models, the pattern-based models and the combined partition models discussed in Chapter 5.

G:	set of grids (functions) that contain variable X
g:	index over G
n_g :	number of segments in the partition for X in grid $g \in G$
i:	index over break points/segments in the partition for X in grid $g \in G$
V^g :	set of vertices in grid $g \in G$
$V^g(i)$	set of vertices in grid g that lie along break point i
$\bar{x}_v^g$:	X-value at vertex v
f_v:	function value at vertex v
$\hat{h}^g$:	approximate function value for grid $g \in G$
α_v^g :	vertex weight variable for vertex $v \in V^g$ in grid $g \in G$
Z_i^g :	binary segment selection variable defined for each segment $i \in I_g$ in grid $g \in G$
λ_i^g :	marginal weight variable for break point i in grid $g \in G$
W_i^g:	binary cumulative segment selection variable for segment i in grid $g \in G$
ω_i^g :	cumulative marginal weight variable for break point i in grid $g \in G$
q:	index over reflected binary codes
B_q^g :	binary variable that is one if we select in the partition for X in grid $g \in G$ a segment that has a one at the q'th position in its gray code
P:	set of partitioning patterns for variable X
p:	index over P

G_p:	set of grids (functions) that contain variable X and use pattern $p \in P$
n_p:	number of break points in partitioning pattern $p \in P$
j:	index over break points/segments in partitioning pattern $p \in P$
S_j^p :	binary segment selection variable for segment *j* in partitioning pattern $p \in P$
σ_j^p:	marginal weight variable for break point *j* in partitioning pattern $p \in P$
T_j^p :	cumulative segment selection (binary) variable for segment *j* in partitioning pattern $p \in P$
τ_j^p:	cumulative marginal weight variable (continuous) for break point *j* in partitioning pattern $p \in P$
C_q^p:	binary variable that is one if we select in partitioning pattern $p \in P$ a segment that has a one at the *q*'th position in its gray code
m:	number of break points in the combined partition for *X*
k:	index over break points and segments in the combined partition for *X*
R_k:	binary segment selection variable defined for segment *k*
ρ_k:	marginal weight variable for break point *k*
M_k :	cumulative segment selection (binary) variable for segment *k* in the combined partition for *X*
μ_k:	cumulative marginal weight variable (continuous) for break point *k* in the combined partition for *X*
D_q:	binary variable that is one if we select in the combined pattern for *X* a segment that has a one at the *q*'th position in its gray code

References

Adhya A, Tawarmalani, M, Sahinidis, NV (1999) A Lagrangian approach to the pooling problem. Ind. Eng. Chem. Res. 38: 1956–1972

Al-Khayyal FA, Falk JE (1983) Jointly constrained biconvex programming. Mathematics of Operations Research 8(2): 273–286

Ahuja RK, Magnanti TL, Orlin JB (1993) Network Flows—Theory, Algorithms and Applications,Prentice Hall, New Jersey.

Ali MM, Khompatraporn C, Zabinsky ZB (2005) A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. Journal of Global Optimization 31: 635–632.

Androulakis IP, Maranas CD, Floudas CA (1995) Alpha bb: A global optimization method for general constrained nonconvex problems. Journal of Global Optimization, 7(4):337–363.

Arnold, VI (1963) On functions of three variables. Givental, AB, ed Vladimir I. Arnold: Collected Works, Volume 1, Representations of Functions, Celestial Mechanics and KAM Theory (Springer), 5–8.

Babayev, DA (1997) Piece-wise linear approximation of functions of two variables. Journal of Heuristics 2: 313–320.

Balakrishnan, A, Graves SC (1989) A composite algorithm for a concave-cost network flow problem. Networks 19: 175–202.

Chien M, Kuh E (1977) Solving nonlinear resistive networks using piecewise-linear analysis and simplicial subdivision. IEEE Trans. Circuits Syst. 24(6): 305–317

Croxton, KL, Gendron B, Magnanti TL (2003) A comparison of mixed-integer programming models for non-convex piecewise linear cost minimization problems. Management Science 49(9): 1268–1273.

Dahl G., Realfsen B (2000) Curve approximation constrained shortest path problems. Networks, 36: 1–8

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

Dantzig, GB (1963) Linear Programming and Extensions. (Princeton University Press, Princeton, NJ).

Domschke P, Geißler B, Kolb O, Lang J, Martin A, Morsi A (2010), Combination of Nonlinear and Linear Optimization of Transient Gas Networks. INFORMS Journal on Computing 23(4): 1–13

Duran, MA and Grossman IE (1986) An outer-approximation algorithm for a class of mixed-integer nonlinear programs Mathematical Programming 36(3), 307-339

Falk, JE and Soland, RM (1969) An algorithm for separable nonconvex programming problems.Management Science 15:550–569

Faria DC, Bagajewicz MJ (2011) Novel bound contraction procedure for global optimization of bilinear MINLP problems with applications to water management problems. Computers and Chemical Engineering 35: 446–455

Floudas, CA, Gounaris, CE (2009) A review of recent advances in global optimization. Journal of global optimization 45(1):3–38

Frey, PJ, George PL (2000) Mesh Generation–Application to finite elements (Hermes Science Publishing, UK) 97–133.

Gilbert, EN (1957) Gray codes and paths on the n-Cube. The Bell System Technical Journal, May 1957.

GonzAlez, JG, Castro GA (2001) Short-term hydro scheduling with cascaded and head-dependent reservoirs based on mixed-integer linear programming. 2001 IEEE Porto Power Tech Conference.

Karuppiah, R., Furman, KC., Grossmann, I E(2008) Global optimization for scheduling refinery crude oil operations. Computers and Chemical Engineering 32 (11), 2745–2766

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

Li, HL, Hu CS (1999) Global optimization method for nonconvex separable programming problems. European Journal of Operational Research 117(2):275–292

Li, HL, Lu HC, Huang CH, Hu NZ (2009) A superior representation method for piecewise linear functions. INFORMS Journal on Computing 21(2):314–321

Linderoth J (2005) A simplicial branch-and-bound algorithm for solving quadratically constrained quadratic programs. Math. Program. Ser. B 103: 251–282

Marnanti TL, Stratilla D (2012) Separable concave optimization approximately equals piecewise linear optimization. Bienstock D, Nemhauser G, eds. Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science Volume 3064, 2004: 234–243

Meyer, CA, Floudas CA (2004) Trilinear monomials with mixed sign domains: Facets of the convex and concave envelopes. J. Glob. Optim. 29(2): 125–155

Meyer, CA, Floudas, CA (2005) Convex envelopes for edge-concave functions. Math. Program. 103(2): 207–224

Misener, R, Floudas CA (2010) Piecewise-linear approximations of multidimensional functions. Journal of Optimization Theory and Applications 145: 120–147

Padberg, M (2000) Approximating separable nonlinear functions via mixed zero-one programs. Oper. Res. Lett. 27: 1–5

Price KV, Storn RM, Lampinen JA (1998) Differential Evolution—A Practical Approach to Global Optimization, Springer, Berlin Heidelberg New York

Radó F (1988) The Euclidean Multifacility Location Problem. Operations Research. 36(3): 485-492

Rovatti, R, Ambrosia CA, Lodi A, Martello S (2014) Optimistic MILP modeling of nonlinear optimization problems. European Journal of Operational Research 239: 32–45

Ryoo, HS and Sahinidis NV (1995) Global optimization of nonconvex nlps and minlps with applications in process design. Computers & Chemical Engineering, 19(5):551–566

Ryoo, HS and Sahinidis NV (1996) A branch-and-reduce approach to global optimization Journal of Global Optimization. 8(2):107–138

Sherali, HD (2001) On mixed-integer zero-one representations for separable lower-semi-continuous piecewise-linear functions. Operations Research Letters 28**:** 155–160.

Laan, GV, Talman AJJ (1980) Simplicial Fixed Point Algorithms, (Mathematisch Centrum 1980, Amsterdam)

Todd, M (1977) Union Jack triangulations. Karamardian S, ed Fixed Points: Algorithms and Applications (Academic Press, New York), 315–336.

Wahlbin LB (1998) General principles of superconvergence in Galerkin finite element methods. Krízek M, Neittaanmäki P, Stenberg R, eds. Finite Element Methods – Superconvergence, Post-processing and a Posteriori Estimates, Lecture Notes in Pure and Applied Mathematics, Volume 196, 2004: 269–285

Westerlund T and Pörn R (2002) Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. Optimization and Engineering, 3:235–280

Westerlund T, Skrifvars H, Harjunkoski I, Pörn R (1998) An extended cutting plane method for a class of non-convex minlp problems. Computers & Chemical Engineering, 22(3):357–365.

Vaidyanathan R and El-Halwagi M (1996). Global optimization of nonconvex minlps by interval analysis. Grossmann IE, ed Global Optimization in Engineering Design, Kluwer Academic Publishers, Dordrecht, 175–193

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

Vielma, JP, Ahmed S, Nemhauser G. (2010) A note on ‘A superior representation method for piecewise linear functions’. INFORMS Journal on Computing 22(3): 493–497.

Vielma, JP, Nemhauser G (2011) Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. Mathematical Programming, Series A. 128: 49–72.

Vielma, JP (2014) Mixed Integer Linear Programming Formulation Techniques (To appear in the Society for Industrial and Applied Mathematics (SIAM)

Wahlbin LB (1998) General principles of superconvergence in Galerkin finite element methods. Krizek M, Neittaanmaki P, Stenberg R (eds) Finite Element Methods, (CRC Press): 269–285

Yousef AA, Gentil P, Jensen JL, Lake LW (2006) A capacitance model to infer interwell connectivity from production and injection rate fluctuations. SPE Reservoir Evaluation and Engineering 9(6):630-646

Zamora JM and Grossmann IE (1999) A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. Journal of Global ptimization, 14:217:249

Zhang, H, Wang S (2008) Linearly constrained global optimization via piecewise-linear approximation. Journal of Computational and Applied Mathematics. 214: 111 – 120