TOPICS IN MIXED INTEGER NONLINEAR PROGRAMMING

by

Kumar Abhishek

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Industrial Engineering

Lehigh University

April 2008

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____
Date

_____
Dr. Jeffrey T. Linderoth
Dissertation Advisor

_____
Accepted Date

Committee:

_____
Dr. Theodore K. Ralphs, Chairman

_____
Dr. Jeffrey T. Linderoth

_____
Dr. Sven Leyffer

_____
Dr. Garth T. Isaak

TO MY PARENTS AND TO VIBHA.

FOR THEIR LOVE, SUPPORT AND ENCOURAGEMENT.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Dr. Jeffrey Linderoth, for all his guidance and help throughout this dissertation, and creating an environment that fostered my personal growth and maturity as a researcher. He has been a constant source of ideas and experienced suggestions all along this period. His patience, when things moved slowly, and his capability to steer my thoughts in productive directions are only a few of the many unforgettable things for which I am immensely thankful for. I could not have hoped for a better advisor.

Special thanks are due to Dr. Sven Leyffer for his guidance, knowledge and enthusiasm in discussing topics relevant to my work in this dissertation. The two summers that I spent at Argonne National Laboratory under his guidance were a great learning experience for me, as well as a lot of fun. He has been a great source of inspiration for me during this time. I would also like to thank the other committee members, Dr. Ted Ralphs, and Dr. Garth Isaak. Their insight, feedback, and constant support was a significant help in this dissertation.

I would like to thank the U.S. Department of Energy, and the National Science Foundation (NSF), for the grants that supported my research. I would also thank the High Performance Computing (HPC) group at Lehigh University for providing me access to the Condor grid and the cluster that enabled my research. I would specially

like to acknowledge my fellow COR@L lab members, Menal and Ashutosh for making our lab a wonderful place to work. Many thanks to my colleagues - Scott, Alex, Gaurav, Zeliha, Mike, Banu, Berrin, Mustafa and Yan for being such great friends and for their moral support. I would like to express my gratitude to Pradeep, and Shweta. The countless coffee and dinner sessions, and inane discussions at unearthly times are just some of the things for which I am thankful to have them as such good friends. Thanks are also due to Ravi, Anil, Navneet, and Shubhra for making my stay at Lehigh very memorable.

Words are not enough to express my gratitude to my Dad and Mom for their love, support, and blessings. I thank them for everything in my life. I deeply acknowledge my kid brother, Abhinav, and my sisters, Manisha and Shweta, for their love and support in every phase of my life. Finally, the love and encouragement of Vibha was of immeasurable comfort during the latter half of this work. It is because of her support, strength, and also her efforts in waking me up everyday, that I might finally finish this dissertation work. You bring so much happiness to me. I would like to thank you all for being the most wonderful family one can ever have. I love you all very much.

Contents

List of Tables

List of Figures

Abstract

A Mixed Integer Nonlinear Program (MINLP) is the problem of minimizing a nonlinear function subject to nonlinear constraints and integrality restrictions on some or all of the decision variables. MINLP is one of the most fundamentally challenging problems in optimization, its difficulty arising from the combination of functional nonlinearities and nonconvexity induced by integrality restrictions. Nevertheless, MINLP is a tremendously important problem, with many important engineering and operations applications being formulated as MINLP models. The main focus of this thesis is on *convex* MINLPs, where the objective function is convex, and the points that satisfy the nonlinear constraints form a convex set. We aim to develop an efficient framework and set of methods for solving convex MINLPs, as well as to demonstrate the advantages of modeling applications as MINLPs.

To achieve this goal, we introduce an algorithmic framework that aims to solve MINLPs at a cost which is a small multiple of the cost of solving comparable mixed integer linear programs. First, we discuss how to implement a (known) linearization-based algorithm (known as LP/NLP branch-and-bound) using existing software components for mixed integer linear programming (MILP) branch-and-cut and (continuous) nonlinear programming. Implementing the algorithm in an existing MILP branch-and-cut framework provides a convenient mechanism for exploring the impact

of various advanced MILP features, such as cutting planes, branching rules, heuristics, and node selection strategies. Further, we discuss new ideas for effectively developing and managing the linear model. We exploit the convexity property of the model to generate linearizations, treating them in a manner similar to the way cutting planes are treated in a branch-and-cut framework for MILPs. Our implementation, called FilMINT, is a powerful and flexible software package for MINLP—competitive with all other known software available for this problem class.

The framework provides us with the means to investigate different heuristic schemes and solution techniques for solving MINLPs. The area of primal heuristics for MINLPs deserves special attention because of the dramatic impact it can have in reducing the tree search. We investigate three heuristics for getting feasible solutions, based on the principle of the feasibility pump. The first scheme is based on solving a nonlinear program and rounding its solution iteratively. The second scheme is an extension of a known heuristic for MINLPs and is based on iteratively solving a nonlinear program and an MILP. We suggest ways to improve the linear model, reduce the time spent in enumeration, and improve an incumbent within this scheme. The third scheme integrates the solution of an NLP with the tree search, similar to the way in which the LP/NLP algorithm avoids the re-solution of the MILP iteratively. The last two approaches are exact solution schemes for convex MINLPs, since they do not cycle. Our computational experience validates the effectiveness of our approach for use in a branch-and-cut framework.

Finally, we illustrate the advantages of using MINLP modeling and solution techniques for an important class of programs, called Mixed Variable Programs (MVP), that are used to model some important applications. A typical MVP model contains

categorical variables that do not allow continuous relaxations, and requires heuristics for its solution. We use integer and nonlinear modeling techniques to formulate such problems and illustrate our approach on a thermal insulation system. We come up with three different MINLP models, with varying degree of smoothness. This allows us to solve such problems using more sophisticated techniques. We highlight the pros and cons of our approach and present numerical results for the model using both FilMINT and a branch-and-bound solver based on nonlinear programming relaxations. Our approach provides a blueprint for modeling such design problems containing categorical variables.

Chapter 1

Introduction

1.1 Motivation

As efforts are made to model real-world systems with ever-increasing accuracy and scale, mathematical models have become increasingly complex. It is often necessary to include binary/integer variables in order to model phenomena such as fixed charges, dichotomies, disjunctive constraints, piecewise linear functions, and yes-no decisions. In order to model a system accurately, it is sometimes necessary to include further complexity in the form of nonlinearity. Nonlinearities can be used to give accurate representations of such phenomena as covariance, economies of scale, or physical properties such as enthalpy and equilibrium. A mathematical model containing integer/binary variables and nonlinear objective and constraints is in general referred to as a Mixed Integer Nonlinear Program (MINLP).

MINLP is one of the most difficult classes of optimization problems. MINLP is NP-hard, which means that no polynomial-time algorithm is known for MINLP. The coupling of integer restrictions with nonlinearities makes the class of MINLPs

challenging from a theoretical, algorithmic, and computational point of view.

A Mixed Integer Linear Program (MILP) can be seen as a special case of MINLP where the objective function and constraints are linear functions. It is well-known that MILP is NP-Hard (see Garey and Johnson [1979]). However, much progress has been made in recent years in our ability to solve practically-sized MILPs. The reader is referred to Schrijver [1986], Nemhauser and Wolsey [1988], and Wolsey [1998] for an in-depth treatment of the theory of mixed-integer linear programming. The reader is also referred to Aardal et al. [2005], and Bertsimas and Weismantel [2005] for reviewing some of the recent progress in solution techniques for MILPs. The most common approach for handling MILPs is the LP-based branch-and-bound implemented in commercial solvers like CPLEX (CPLEX Optimization), XPRESS (Laundy [1998]), or non-commercial solvers like MINTO (see Nemhauser et al. [1994]) and SYMPHONY (see Ralphs and Ladányi [2000]). The Computational Infrastructure for Operations Research (COIN-OR), provides a useful collection of open-source software components that can be used in a branch-and-cut framework. Much of the progress can be attributed to improvements in cutting plane and LP-based branch-and-bound technologies (see Barnhart et al. [1998], Balas et al. [1993], and Johnson et al. [2000]).

There has also been steady progress over the years in the development and successful implementation of algorithms for Nonlinear Programs (NLP), which is the problem of minimizing a nonlinear objective function subject to nonlinear constraints over a domain of continuous-valued variables. We refer the reader to Bazaraa et al. [1993], and Nocedal and Wright [1999] for a detailed treatment of nonlinear programming techniques. The developments in this field have been mostly concentrated on local optimization methods guaranteeing global optimality under certain convexity

assumptions. However, global optimization of constrained nonlinear programming problems without the assumption of convexity is an NP-hard problem in general (see Pardalos and Vavasis [1992], Vavasis [1991]). With the advances made in improving the computational schemes for convex nonlinear programming, and with the advent of NLP solvers like filterSQP (Fletcher and Leyffer [1998b]) and IPOPT (Wächter and Biegler [2006]), much larger problems can now be solved in reasonable time.

While the field of mixed-integer linear programming and nonlinear programming have progressed at a considerable pace, relatively less progress has been made on solution methodologies for MINLP. With a modern, state-of-the-art MILP solver, it is possible to solve models with millions of variables and constraints, whereas the dimension of solvable convex MINLPs is often limited by a number that is smaller by three or four orders of magnitude. This has motivated us to look at MINLPs and their solution methodologies, with the goal of developing a better framework and set of methods for solving convex MINLPs. Improvements in the solution methodologies for MINLP also motivates us to expand the domain of real-life applications that can be modeled and solved as MINLPs.

Our end goal is to be able to improve solution techniques to the point that MINLPs can be solved at a cost which is a small multiple of the cost of solving a comparable MILP. We believe that the progress made in reducing the current gap in theoretical and computational schemes for MINLP, and in increasing the domain of MINLP applications will contribute towards making mixed-integer nonlinear programming a practical and sophisticated mathematical tool for solving challenging real-life applications.

1.2 Introduction to the problem

A MINLP is conveniently expressed as

$$\begin{aligned} z_{\text{MINLP}} = \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & g(x,y) \leq 0, \\ & x \in X, y \in Y \cap \mathbb{Z}^p, \end{aligned} \tag{MINLP}$$

where $f : \mathbb{R}^{n\times p} \to \mathbb{R}$ and $g : \mathbb{R}^{n\times p} \to \mathbb{R}^m$ are twice continuously differentiable functions, x and y are continuous and discrete variables, respectively, X is a polyhedral subset of $\mathbb{R}^n$, and Y is a bounded polyhedral subset of $\mathbb{Z}^p$. In this thesis, we focus on the case where the functions f and g_j are convex, so that by relaxing the restriction $y \in \mathbb{Z}^p$, a convex program (minimization of a convex function over a convex set) is formed. Many of the techniques suggested may be applied as a heuristic in the case that one or more of the functions are nonconvex, implying that there is no guarantee of global optimality.

1.3 Applications of MINLP

Applications for MINLPs arise in many important application areas: for example chemical engineering, energy generation and utilities, communication and transportation networks, and facility planning. Chemical engineering applications of interest include the design of batch plants (Grossmann and Sargent [1979], Kocis and Grossmann [1988]), process synthesis (Kocis and Grossmann [1988], Floudas [1995]), cyclic scheduling (Jain and Grossmann [1998]), design of distillation columns (Viswanathan and Grossmann [1993]), and pump configuration optimization (Westerlund et al.

[1994]). Energy generation and utilities related applications include nuclear core reload optimization (Quist et al. [1998], Allaire and Castro [2001]) and the electric unit commitment problem (Sheble and Fahd [1994]). An example of transportation network related application is the gas transmission problem (De Wolf and Smeers [2000]). One can refer to Dembo et al. [1989] for a survey on applications of nonlinear network models. An application of facility layout and planning problem includes the block layout design problem (Castillo et al. [2005], Sawaya et al. [2006]). Another application of interest includes the trimloss problem (Harjunkoski et al. [1988]) in the paper industry. A number of these problems have been modeled in practice as convex MINLPs. However, we will show that some of these problems that are modeled as nonconvex MINLPs can be transformed to convex MINLPs, thus making it possible to solve these problems to provable optimality. A brief description of some of these problems is given next.

1.3.1 Design of Batch Plants

In this problem of design of multiproduct batch plants (Grossmann and Sargent [1979], Kocis and Grossmann [1988]), it is assumed that the plant consists of M batch processing stages where fixed amounts Q_i of N products must be manufactured. At each stage j, p_j units operate independently in parallel and all the units within a given stage j have the same size v_j. This is illustrated in Figure 1.1. The design problem consists of determining for each stage j, the number of parallel units p_j and their sizes v_j, and for each product i, the batch sizes b_i and their cycle times t_i. Data for the problem are the horizon time H, cost coefficients α_j, cost exponents β_j for the units, size factors S_{ij}, and processing times T_{ij} each each product i at stage j.

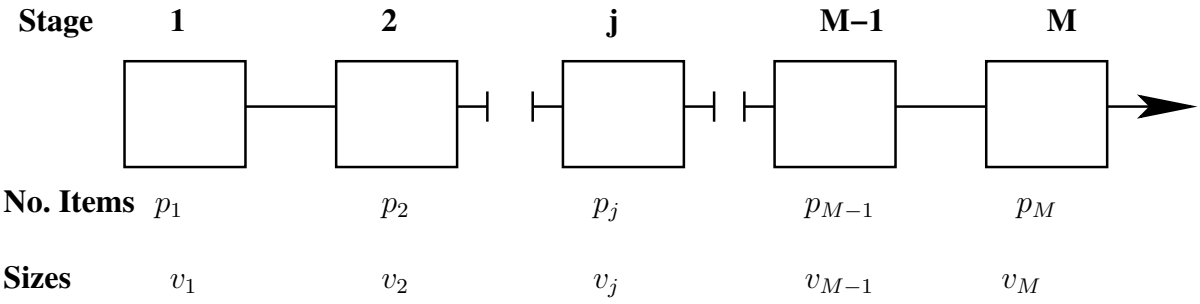

Figure 1.1: Sequential multiproduct batch plant

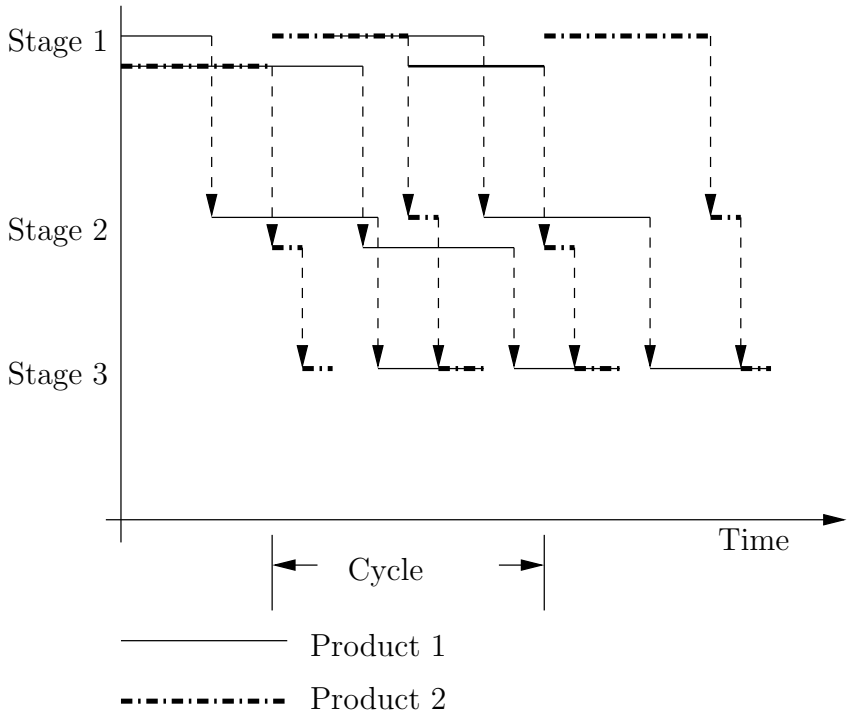

Figure 1.2: Scheduling of a three-stage batch plant for two products.

The required unit size, v_{ij}, for processing product i in stage j is given by

$$v_{ij} = b_i S_{ij} \quad i = 1 \ldots N,\ j = 1 \ldots M,$$

where the size factor $S_{ij} > 0$ is a constant. The unit sizes v_{ij} must then be chosen to satisfy

$$v_j \geq v_{ij} \quad i = 1 \ldots N,\ j = 1 \ldots M.$$

The plant operates in a steady cyclic condition, and no auxiliary storage is available, so each product batch, once started, must be processed through all stages without any waiting time. The situation is illustrated in Figure 1.2 for two products.

Over a given period of operation H, the number of batches, n_i, for each product, and their sizes b_i, must be chosen to achieve the required production Q_i, $Q_i = n_i b_i$. The number of units in each stage must be chosen so that all the batches can be processed. On an average, the time taken to process the n_i batches of product i in stage j will be $(n_i T_{ij}/p_j)$, and thus we must have

$$H \geq \sum_{i=1}^{N} n_i T_{ij}/p_j \quad j = 1 \ldots M$$

It follows that scheduling can always be done if the period H is long enough to allow the maximum cycle time, t_i, given by

$$t_i = \max_{j=1\ldots M} (T_{ij}/p_j) \quad i = 1 \ldots N$$

in every stage. This observation leads to the horizon constraint

$$H \geq \sum_{i=1}^{N} n_i t_i = \sum_{i=1}^{N} \frac{Q_i}{b_i} t_i$$

Garfinkel and Nemhauser [1972] use the binary (base 2) number system to show that any integer $a \in [0, 2^R - 1]$ can be represented by a sequence of R bits given as $y_{R-1}, y_{R-2}, \ldots, y_i, \ldots, y_0$, where any bit $y_i \in \{0, 1\}$ represents the value 2^i (in decimal number system) if y_i is set to 1, and 0 otherwise. Given any binary sequence, an integer a can therefore be represented as $a = \sum_{i=0}^{R-1} 2^i y_i$. Grossmann and Sargent [1979] use this fact to represent the number of parallel units p_j, in terms of 0-1 variables (y_{kj}) given below:

$$p_j = 1 + \sum_{k=1}^{R} 2^{k-1} y_{kj} \quad j = 1 \ldots M$$

For example, if a maximum of four units is considered for stage j, then p_j can be expressed as $p_j = 1 + y_{1j} + 2y_{2j}$ $(R = 2)$.

The optimal design of multi-product batch plants can now be formulated as the following MINLP (MIPB1):

$$z = \min \sum_{j=1}^{M} \alpha_j p_j v_j^{\beta_j} \qquad \text{(investment cost)}$$

subject to:

$$v_j \geq S_{ij} b_i \quad i = 1 \ldots N, \quad j = 1 \ldots M \qquad \text{(size for stage j)}$$

$$p_j t_i \geq T_{ij} \quad i = 1 \ldots N, \quad j = 1 \ldots M \qquad \text{(cycle time for product i)}$$

$$\sum_{i=1}^{N} \frac{Q_i t_i}{b_i} \leq H \qquad \text{(horizon constraint)}$$

$$p_j = 1 + \sum_{k=1}^{R} 2^{k-1} y_{kj} \quad j = 1 \ldots M \qquad \text{(number of parallel units)}$$

$$1 \le p_j \le p_j^U \quad j = 1 \ldots M \qquad \text{(bounds)}$$

$$v_j^L \le v_j \le v_j^U \quad j = 1 \ldots M$$

$$t_i^L \le t_i \le t_i^U \quad i = 1 \ldots N$$

$$b_i^L \le b_i \le b_i^U \quad i = 1 \ldots N$$

$$y_{kj} \in \{0, 1\} \quad k = 1 \ldots R, \quad j = 1 \ldots M.$$

where p_j^U, v_j^L, and v_j^U are specified bounds, while valid bounds for t_i and b_i can be determined as follows:

$$t_i^L = \max_j \{T_{ij}/p_j^U\}, \quad t_i^U = \max_j \{T_{ij}\},$$
$$b_i^L = \frac{Q_i}{H} \max_j \{\frac{T_{ij}}{p_j^U}\}, \quad b_i^U = \min\{Q_i, \min_j \frac{v_j^U}{S_{ij}}\}.$$

In (MIPB1), the inequality constraints for volumes and the equations for the number of parallel units are linear, but the rest of the model involves nonlinear functions. We note that the horizon time constraint and the objective function are nonconvex, and the nonlinear inequalities for the cycle times are quasi-convex. Recall that a function $f : S \to \mathbb{R}$ defined on a convex subset $S \subseteq \mathbb{R}^n$ is defined to be quasi-convex, if for any $x, y \in S$ and $\lambda \in [0, 1]$, we have the relation

$$f(\lambda x + (1 - \lambda)y) \le \max\{f(x), f(y)\}.$$

Through logarithmic transformations, the above formulation can be modeled as a convex MINLP. This requires the definitions of the transformed variables $\bar{v}_j = \ln[v_j]$,

$\bar{p}_j = \ln[p_j]$, $\bar{b}_i = \ln[b_i]$, and $\bar{t}_i = \ln[t_i]$. These transformations yield the following convex formulation (MIPB2):

$$z = \min \sum_{j=1}^{M} \alpha_j \exp[\bar{p}_j + \beta_j \bar{v}_j] \qquad \text{(investment cost)}$$

subject to:

$$\bar{v}_j \geq \ln[S_{ij}] + \bar{b}_i \quad i = 1 \ldots N, j = 1 \ldots M \qquad \text{(size for stage j)}$$

$$\bar{p}_j + \bar{t}_i \geq \ln[T_{ij}] \quad i = 1 \ldots N, \quad j = 1 \ldots M \qquad \text{(cycle time for product i)}$$

$$\sum_{i=1}^{N} Q_i \exp[\bar{t}_i - \bar{b}_i] \leq H \qquad \text{(horizon constraint)}$$

$$\bar{p}_j = 1 + \sum_{k=1}^{p_j^U} \ln[k] y_{kj} \quad j = 1 \ldots M \qquad \text{(number of parallel units)}$$

$$\sum_{k=1}^{p_j^U} y_{kj} = 1 \quad j = 1 \ldots M \qquad \text{(only 1 choice of } \bar{p}_j\text{)}$$

$$0 \leq \bar{p}_j \leq \ln[p_j^U] \quad j = 1 \ldots M \qquad \text{(bounds)}$$

$$\ln[v_j^L] \leq \bar{v}_j \leq \ln[v_j^U] \quad j = 1 \ldots M$$

$$\ln[t_i^L] \leq \bar{t}_i \leq \ln[t_i^U] \quad i = 1 \ldots N$$

$$\ln[b_i^L] \leq \bar{b}_i \leq \ln[b_i^U] \quad i = 1 \ldots N$$

$$y_{kj} \in \{0, 1\} \quad k = 1 \ldots p_j^U, j = 1 \ldots M.$$

Note that in this formulation, all nonconvexities have been eliminated. The nonlinearities in the model appear only in the objective function and the horizon constraint, and in both cases, the exponential functions are convex.

1.3.2 Block Layout Design Problem

The block layout design problem (BLDP) with unequal areas (Castillo et al. [2005]) is a fundamental optimization problem encountered in many manufacturing and service organizations. The problem is concerned with finding the most efficient arrangement of a given number of departments with unequal area requirements within a facility.

In BLDP, there are N departments, and the floor area of the facility is a rectangle of size $w_F \times h_F$. For each department i, denote its position by the coordinates of its center by (x_i, y_i) and its width and height dimensions by w_i and h_i respectively. Thus, the decision variables for determining the department location and size are given by (x_i, y_i, w_i, h_i). Valid lower and upper bounds on w_i and h_i are denoted by $(w_i^{\text{low}}, w_i^{\text{up}})$ and $(h_i^{\text{low}}, h_i^{\text{up}})$, respectively. Each department i is required to have an area a_i and the relationship (material-handling flow or preference regarding adjacency) c_{ij} between departments i and j is given.

Nonlinearities in the problem arise because of the area constraints. The integrality restrictions in the model arise to prevent departments from overlapping with each other. The area requirement for each department i can be written as $w_i h_i = a_i$. This is a nonconvex, hyperbolic constraint. The area constraint is illustrated by Figure 1.3. To prevent departments from overlapping, one needs to ensure that the departments may overlap at most with only one of the axes(x or y). This may be done by the condition:

$$|y_i - y_j| - \frac{1}{2}(h_i + h_j) < 0 \Rightarrow |x_i - x_j| - \frac{1}{2}(w_i + w_j) \geq 0 \quad \forall 1 \leq i < j \leq N,$$
$$|x_i - x_j| - \frac{1}{2}(w_i + w_j) < 0 \Rightarrow |y_i - y_j| - \frac{1}{2}(h_i + h_j) \geq 0 \quad \forall 1 \leq i < j \leq N.$$

Figure 1.4 illustrates this condition.

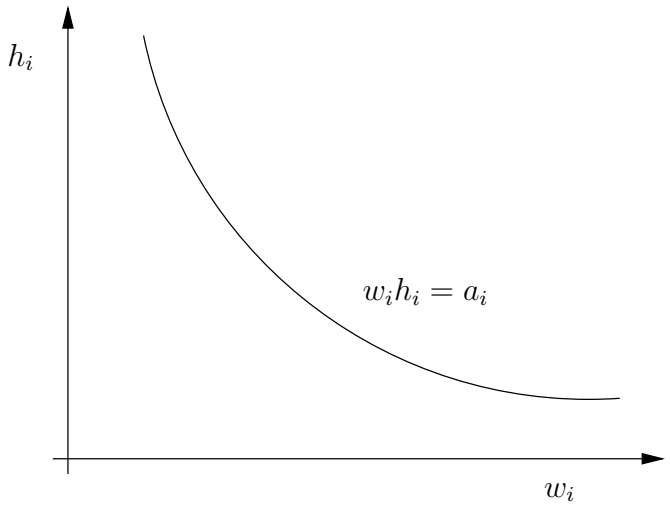

Figure 1.3: Department area constraint

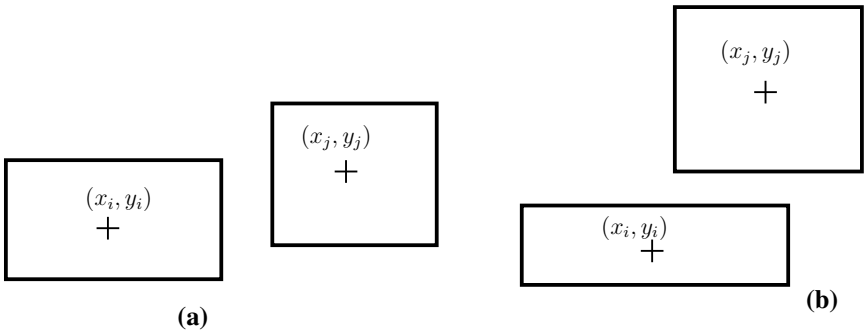

Figure 1.4: Separation Illustration

These conditions can be enforced with the help of binary variables X_{ij} and Y_{ij}. The binary variables X_{ij} are used to indicate the axis about which the overlap prevention constraint is active. $X_{ij} = 1$ implies overlap prevention along the y-axis is enforced, and $X_{ij} = 0$ implies overlap prevention along the x-axis is active. The binary variables Y_{ij} are used to indicate the relative positions of departments i and j along the axis where overlap prevention constraints are active. Thus, $Y_{ij} = 1$ implies that department j lies "after" department i, and $Y_{ij} = 0$ implies that department i lies "after" department j. The constraints that are used to enforce overlap prevention are now given as:

$$\begin{aligned}
\frac{1}{2}(h_i + h_j) - (y_i - y_j) &\leq h_F^{\text{up}}(1 - X_{ij} + Y_{ij}) \qquad \forall 1 \leq i < j \leq N, \\
\frac{1}{2}(h_i + h_j) - (y_j - y_i) &\leq h_F^{\text{up}}(2 - X_{ij} - Y_{ij}) \qquad \forall 1 \leq i < j \leq N, \\
\frac{1}{2}(w_i + w_j) - (x_i - x_j) &\leq w_F^{\text{up}}(X_{ij} + Y_{ij}) \qquad \forall 1 \leq i < j \leq N, \\
\frac{1}{2}(w_i + w_j) - (x_j - x_i) &\leq w_F^{\text{up}}(1 + X_{ij} - Y_{ij}) \qquad \forall 1 \leq i < j \leq N.
\end{aligned}$$

The above constraints, along with the integrality of X_{ij} and Y_{ij}, ensure that at least one of the above four constraints are binding. This can be easily seen since there are only four different combinations that can be obtained using X_{ij} and Y_{ij}, and each of those combinations causes the right hand side of exactly one of the inequalities to become zero.

The model (BLDP1) can now be formulated as follows:

$$\min \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij}(|x_i - x_j| + |y_i - y_j|) \tag{1.3.1}$$

subject to:

$$y_i - \frac{1}{2}h_i \geq 0 \qquad \forall i = 1 \dots N \tag{1.3.2}$$

$$y_i + \frac{1}{2}h_i \leq h_F \qquad \forall i = 1 \dots N \tag{1.3.3}$$

$$x_i + \frac{1}{2}w_i \leq w_F \qquad \forall i = 1 \dots N \tag{1.3.4}$$

$$x_i - \frac{1}{2}w_i \geq 0 \qquad \forall i = 1 \dots N \tag{1.3.5}$$

$$h_i^{\text{low}} \leq h_i \leq h_i^{\text{up}} \qquad \forall i = 1 \dots N \tag{1.3.6}$$

$$w_i^{\text{low}} \leq w_i \leq w_i^{\text{up}} \qquad \forall i = 1 \dots N \tag{1.3.7}$$

$$w_F^{\text{low}} \leq w_F \leq w_F^{\text{up}} \tag{1.3.8}$$

$$h_F^{\text{low}} \leq h_F \leq h_F^{\text{up}} \tag{1.3.9}$$

$$h_i w_i - a_i = 0 \qquad i = 1 \dots N \tag{1.3.10}$$

$$\frac{1}{2}(h_i + h_j) - (y_i - y_j) \leq h_F^{\text{up}}(1 - X_{ij} + Y_{ij}) \qquad \forall 1 \leq i < j \leq N \tag{1.3.11}$$

$$\frac{1}{2}(h_i + h_j) - (y_j - y_i) \leq h_F^{\text{up}}(2 - X_{ij} - Y_{ij}) \qquad \forall 1 \leq i < j \leq N \tag{1.3.12}$$

$$\frac{1}{2}(w_i + w_j) - (x_i - x_j) \leq w_F^{\text{up}}(X_{ij} + Y_{ij}) \qquad \forall 1 \leq i < j \leq N \tag{1.3.13}$$

$$\frac{1}{2}(w_i + w_j) - (x_j - x_i) \leq w_F^{\text{up}}(1 + X_{ij} - Y_{ij}) \qquad \forall 1 \leq i < j \leq N \tag{1.3.14}$$

$$X_{ij} \in \{0, 1\} \qquad \forall 1 \leq i < j \leq N \tag{1.3.15}$$

$$Y_{ij} \in \{0, 1\} \qquad \forall 1 \leq i < j \leq N. \tag{1.3.16}$$

(1.3.2)-(1.3.5) ensure that all departments are placed within the facility. (1.3.6)-(1.3.9) deal with bounds on the height of departments and on the width and height of the floor. (1.3.10) deals with the area requirements of each department. (1.3.11)-(1.3.14) ensure that no two departments in the layout overlap. (1.3.15)-(1.3.16) are used to ensure that only one of the constraints in (1.3.11)-(1.3.14) is binding.

We note that it is valid to write the area constraint, $w_i h_i = a_i$, as $w_i h_i \geq a_i$. This constraint is Second Order Cone (SOC) representable, and directly provides a convex reformulation. However, we next present a convex reformulation of this problem by using variable transformations, as suggested by Castillo et al. [2005].

Using transformation $t_i = 1/w_i$, a convex version of the model (BLDP2) may be formulated as follows:

$$\min \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij}(|x_i - x_j| + |y_i - y_j|) \tag{1.3.17}$$

subject to:

$$y_i - \frac{1}{2}h_i \geq 0 \qquad \forall i = 1 \dots N \tag{1.3.18}$$

$$y_i + \frac{1}{2}h_i \leq h_F \qquad \forall i = 1 \dots N \tag{1.3.19}$$

$$x_i + \frac{1}{2t_i} \leq w_F \qquad \forall i = 1 \dots N \tag{1.3.20}$$

$$x_i - \frac{1}{2t_i} \geq 0 \qquad \forall i = 1 \dots N \tag{1.3.21}$$

$$h_i^{\text{low}} \leq h_i \leq h_i^{\text{up}} \qquad \forall i = 1 \dots N \tag{1.3.22}$$

$$w_F^{\text{low}} \leq w_F \leq w_F^{\text{up}} \tag{1.3.23}$$

$$h_F^{\text{low}} \leq h_F \leq h_F^{\text{up}} \tag{1.3.24}$$

$$h_i - a_i t_i = 0 \qquad i = 1 \dots N \tag{1.3.25}$$

$$\frac{1}{2}(h_i + h_j) - (y_i - y_j) \leq h_F^{\text{up}}(1 - X_{ij} + Y_{ij}) \qquad \forall 1 \leq i < j \leq N \tag{1.3.26}$$

$$\frac{1}{2}(h_i + h_j) - (y_j - y_i) \leq h_F^{\text{up}}(2 - X_{ij} - Y_{ij}) \qquad \forall 1 \leq i < j \leq N \tag{1.3.27}$$

$$\frac{1}{2}(\frac{1}{t_i} + \frac{1}{t_j}) - (x_i - x_j) \leq w_F^{\text{up}}(X_{ij} + Y_{ij}) \qquad \forall 1 \leq i < j \leq N \tag{1.3.28}$$

$$\frac{1}{2}(\frac{1}{t_i} + \frac{1}{t_j}) - (x_j - x_i) \leq w_F^{\text{up}}(1 + X_{ij} - Y_{ij}) \qquad \forall 1 \leq i < j \leq N \tag{1.3.29}$$

$$\frac{1}{w_i^{\text{up}}} \leq t_i \leq \frac{1}{w_i^{\text{low}}} \qquad \forall i = 1 \dots N \tag{1.3.30}$$

$$X_{ij} \in \{0,1\}, \qquad \forall 1 \leq i < j \leq N \tag{1.3.31}$$

$$Y_{ij} \in \{0,1\}, \qquad \forall 1 \leq i < j \leq N \tag{1.3.32}$$

1.4 Survey of Existing Solution Methods

Solution strategies explicitly addressing MINLPs have appeared only intermittently in the literature. Much of the earlier work in this field dealt with MINLPs containing nonlinearities restricted to quadratic functions (McBride and Yormark [1980], Gupta and Ravindran [1985], Borchers and Mitchell [1994], Borchers and Mitchell [1997], Bienstock [1996]). These implementations, whenever reported, were restrictive in scope and focus attention on special subclasses of MINLPs.

Methods for solving MINLPs fall into two broad classes. The first class is that of deterministic methods, which, given enough time and provided the problem satisfies certain conditions, such as convexity, terminate with a guaranteed optimal solution or an indication that the problem is infeasible. A number of deterministic methods perform an exhaustive tree search with rules that enable them to intelligently limit the search. The second class of methods are known as heuristic methods. These methods do not provide the guarantee that on termination, the incumbent is a minimizer. Methods that are specifically designed to solve convex programs to optimality can be used as a heuristic for nonconvex programs. Some of the deterministic methods/approaches include Branch-and-Bound (Dakin [1965], Gupta and Ravindran [1985]), Benders Decomposition (Geoffrion [1972]), Outer Approximation (Duran and Grossmann [1986]), Extended Cutting Plane method (Westerlund and Pettersson [1995]), LP/NLP-based Branch-and-Bound (Quesada and Grossmann [1992]), and Integrated SQP with branch-and-bound (Leyffer [2001]). Each of these methods have

their share of advantages and disadvantages with respect to solving convex MINLPs. See Grossmann [2002] for a review of the various deterministic methods for MINLP.

We focus our attention on deterministic methods since we are interested in the solution methodologies for convex MINLPs. Before going into the details of some of these methods, we next define the following related subproblems that are relevant for the solution methodologies for MINLP. We will then go for an in-depth survey of some of these solution methods.

1.4.1 NLP Subproblems

Recall from Section 1.2 that a convex MINLP is expressed as:

$$\begin{aligned} z_{\text{MINLP}} = \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & g(x,y) \leq 0, \\ & x \in X, y \in Y \cap \mathbb{Z}^p, \end{aligned} \tag{MINLP}$$

where $f : \mathbb{R}^{n\times p} \to \mathbb{R}$ and $g : \mathbb{R}^{n\times p} \to \mathbb{R}^m$ are twice continuously differentiable convex functions, and x and y are continuous and discrete variables.

The notation used in the definition of (MINLP), NLP subproblems and the MILPs defined later in this section can be primarily attributed to a paper by Grossmann [2002]. We characterize a node $n \equiv (l,u)$ of the branch-and-bound search tree by the bounds $\{(l,u)\}$ enforced on the integer variables y. Given bounds (l,u) on y, we define the NLP relaxation as

$$\begin{aligned} z_{\text{NLPR}(l,u)} = \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & g(x,y) \leq 0, \end{aligned} \tag{NLPR(l, u))}$$

$$l \leq y \leq u,$$
$$x \in X, y \in Y.$$

If l and u are lower and upper bounds on the y variables for the original instance, then the optimal objective function value $z_{\text{NLPR}(l,u)}$ of (NLPR(l, u)) provides a lower bound on (MINLP); otherwise it provides a lower bound for the subtree whose parent node is $\{(l, u)\}$. In general, the solution to (NLPR(l, u)) may yield nonintegral values for the integer decision variables y.

The NLP subproblem obtained for a fixed y (say y^k), known as (NLP(y^k)), is defined as follows:

$$\begin{aligned} z_{\text{NLP}(y^k)} = \text{minimize} \quad & f(x, y^k) \\ \text{subject to} \quad & g(x, y^k) \leq 0, \\ & x \in X. \end{aligned} \qquad (\text{NLP}(y^k))$$

The value $z_{\text{NLP}(y^k)}$ provides an upper bound to the problem (MINLP) provided (NLP(y^k)) has a feasible solution. When (NLP(y^k)) is infeasible, we consider the feasibility problem for fixed y^k, denoted (NLPF(y^k)$_\infty$):

$$\begin{aligned} z_{\text{NLPF}(y^k)_\infty} = \text{minimize} \quad & u \\ \text{s.t.} \quad & g(x, y^k) \leq u, \\ & x \in X, u \in \mathbb{R}^1. \end{aligned} \qquad (\text{NLPF}(y^k)_\infty)$$

This may be interpreted as the minimization of the ℓ_∞ measure of infeasibility of the

corresponding NLP subproblem (NLP(y^k)). Some other methods deal with maintaining feasibility in any constraint residual once it has been established. For example, one may solve the following problem, which measures the ℓ_1-norm of constraint violation:

$$\begin{aligned} z = \text{minimize} \quad & \sum_{j \in J^-} g_j(x, y^k)^+ \\ \text{s.t.} \quad & g_j(x, y^k) \leq 0, \quad j \in J^+, \\ & x \in X, \end{aligned}$$

where J^+ is the set of constraints that are currently feasible, and J^- is its complement ($J^+ \cap J^- = \emptyset, J^+ \cup J^- = \{1, 2, \ldots, m\}$). Here, $g_j(x, y^k)^+ = \max\{0, g_j(x, y^k)\}$ measures the violation of the nonlinear constraints.

The choice of the feasibility problem to be solved depends on the NLP solver that is used to solve (NLP(y^k)). Some NLP solvers like filterSQP detect infeasibility of (NLP(y^k)) and return the solution to a feasibility problem, which minimizes the scaled ℓ_1-norm of the constraint violation. The form of the feasibility problem that is solved by filterSQP is given as (NLPF(y^k)).

$$\begin{aligned} z_{\text{NLPF}(y^k)} = \text{minimize} \quad & \sum_{j=1}^{m} w_j g_j(x, y^k)^+ \\ \text{s.t.} \quad & x \in X, \end{aligned} \qquad (\text{NLPF}(y^k))$$

for some weights $w_j \geq 0$, not all zero. Specifically, $w_j = 1$ if the constraint is violated, and $w_j = \lambda_j$ (Lagrange multiplier associated with the constraint) otherwise.

From the solution to (NLP(y^k)) or (NLPF(y^k)), we can derive outer approximations for (MINLP). The convexity of the nonlinear functions imply that the linearizations about any point (x^k, y^k) (obtained by solving (NLP(y^k))) form an outer approximation of the feasible set and underestimate the objective function. Specifically, if we introduce an auxiliary variable η in order to replace the objective by a constraint, changing the objective to minimize η and adding the constraint $\eta \geq f(x, y)$, then the $m + 1$ inequalities (OA(x^k, y^k)) are valid for MINLP:

$$f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq \eta,$$

$$g(x^k, y^k) + \nabla g(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0, \qquad \text{(OA}(x^k, y^k)\text{)}$$

where $\nabla g(x^k, y^k)^T = [\nabla g_1(x^k, y^k) : \ldots : \nabla g_m(x^k, y^k)]^T$ is the Jacobian of the nonlinear constraints.

When (NLP(y^k)) is infeasible, valid linearizations can be obtained by solving the feasibility problem (NLPF(y^k)). The linearizations in this case is given by (FC(x^k, y^k)).

$$g(x^k, y^k) + \nabla g(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0. \qquad \text{(FC}(x^k, y^k)\text{)}$$

Figures 1.5 and 1.6 illustrate the geometric interpretation of the linearizations.

We next describe some well-known algorithms for solving MINLPs.

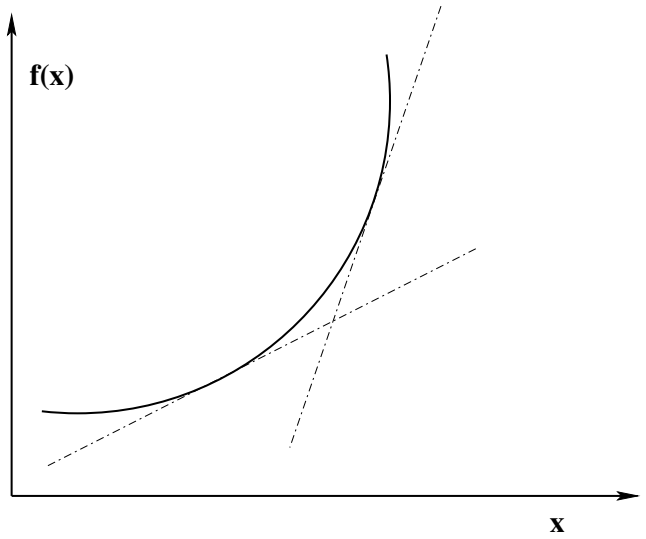

Figure 1.5: Underestimation of the convex objective function

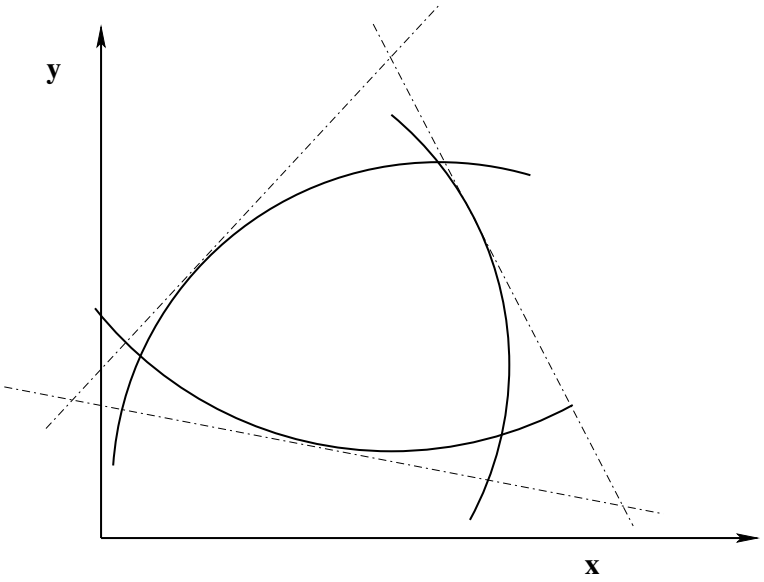

Figure 1.6: Overestimation of the convex feasible region

1.4.2 Branch and Bound

Branch-and-Bound is a classical algorithm for solving discrete optimization problems. The main idea behind branch-and-bound is to solve continuous relaxations of the original problem and to divide the feasible region, eliminating the fractional solution of the relaxed problem. Continuing in this manner yields a tree of problems which is searched for the integer optimum. The first branch-and-bound algorithm for MILP problems was proposed by Land and Doig [1960]. Dakin [1965] was the first to realize that this scheme does not require linearity of the problem functions. To the best of our knowledge, McBride and Yormark [1980] were the first to develop a branch-and-bound algorithm for 0-1 quadratic programming that computes lower bounds by solving the continuous relaxation. Gupta and Ravindran [1985] suggested an implementation of branch-and-bound for convex MINLPs and investigated the effects of problem size on solution time and different branching and node selection strategies. They concluded that the branching rule that branches on the most fractional variable works the best. Nabar and Schrage [1990] also investigated the use of branch-and-bound for solving MINLPs.

We next give a description of the branch-and-bound algorithm for solving (MINLP). The continuous relaxation of (MINLP) at a node $n \equiv (l, u)$ is the NLP subproblem stated before as (NLPR(l, u)). The branch-and-bound algorithm starts by solving the continuous relaxation, (NLPR(l, u)), at the root node. The solution of the continuous relaxation may be denoted as $(\bar{x}, \bar{y})$ and the objective function value as $z_{\text{NLPR}(l,u)}$. If all the discrete variables take integer values, the search is stopped. Otherwise, a tree search is performed in the space of the integer variables, $y_i \in Y$ by changing bounds (l, u) on y variables. The bounds on integer variables are successively tightened at the

corresponding nodes of the tree, giving rise to relaxed NLP subproblems (NLPR(l, u)) mentioned before. At a particular node, (NLPR(l, u)) provides a valid lower bound for the subproblems in the descendant nodes. The tree search is done by branching on an integer variable y_i that has a fractional value $\bar{y}_i$. This is done by introducing two new subproblems from the current node by adding a simple bound:

$$y_i \geq \lceil \bar{y}_i \rceil \tag{1.4.33}$$

to one subproblem and

$$y_i \leq \lfloor \bar{y}_i \rfloor \tag{1.4.34}$$

to the other subproblem. The solution of (MINLP) is contained in the union of the feasible region of the two subproblems and the process can be repeated. The new nodes are termed as *child nodes*. The subtree rooted at the child node corresponding to the branching constraint (1.4.34) is referred to as the down-branch of the parent node, and the subtree rooted at the other child node is referred to as the up-branch. If (NLPR(l, u)) yields an integral solution, then $z_{\text{NLPR}(l,u)}$ provides an upper bound for (MINLP). The tree search continues until there are no pending nodes to be evaluated. The success of the branch-and-bound methodology is largely due to the fact that whole subtrees may be excluded from further consideration by fathoming their respective root node. Fathoming of the node and its subtree occurs when the lower bound obtained by solving (NLPR(l, u)) exceeds the current upper bound for the problem, or when the subproblem is infeasible, or when the node gives a feasible integral solution. Also, fathoming of a node can be done when the NLP solver fails due to numerical

errors, for example, in the case of IEEE errors in function evaluations. Further, NLP solvers may also fail when either the iteration limit is reached, or the trust-region radius (when using trust-region methods for solving NLPs) becomes zero due to the failure of Linear Independence Constraint Qualification (LICQ) or the Mangasarian-Fromovitz Constraint Qualification (MFCQ). In these cases, a MINLP solver should continue branching on such nodes. The error is generally handled subsequently since the child nodes become infeasible in most cases.

Algorithm 1 gives a detailed description of the branch-and-bound algorithm. As previously mentioned, in the description of the algorithm, a node in the tree search is denoted by $n \equiv (l, u)$, where l is the vector of lower bounds and u is the vector of upper bounds on the integer variables y. We denote the parent of the node n by $\rho(n)$. UB denotes the best upper bound obtained at any point of time, and $Z^n \equiv z_{\text{NLPR}(l,u)}$ denotes the lower bound for a sub-tree rooted at node n.

The tree search in a branch-and-bound procedure can be limited by employing effective branching variable selection and node selection strategies. We now assume $y_i \in \{0, 1\}$ for the sake of simplicity. Let y^* be the value of integer variables y at the current node, and let the objective function value of the current node be given by Z^*. Some of the strategies for variable selection for branching include the following:

1. Maximal fractional branching. This implies branching on the variable that is the most fractional. In other words, finding y_i which is farthest from its nearest integer value. This is by computing

$$\min_i |y_i^* - 0.5|.$$

2. Use of pseudocosts. This rule is attributed to the work of Benichou et al. [1971].

Initialization: Set up the continuous relaxation (NLPR(l, u)) of (MINLP).
$UB := \infty$, List of open nodes $R = \{n \equiv (l, u)\}$.
Step 1: If $R = \emptyset$, **then** STOP.
Select a node (say $\{n\}$) from R (node selection).
Step 2: Solve the NLP relaxation (NLPR(l, u)) for the current node $\{n\}$.
if (NLPR(l, u)) infeasible **then**
  Fathom node ($R = R\backslash\{n\}$). Goto Step 1.
else
  Solution is $(\hat{x}, \hat{y})$, objective function value is $Z^n = z_{\text{NLPR}(l,u)}$.
end if
if $(\hat{x}, \hat{y})$ fractional and $Z^n < UB$ **then**
  Select a fractional variable $y_b \notin \mathbb{Z}$. (Branching variable selection)
  Branch by creating two subproblems p and q such that

$$\begin{aligned} \hat{u}_b &\leftarrow \lfloor y_b^k \rfloor, \qquad \hat{u}_j \leftarrow u_j \quad \forall j \neq b \\ \hat{l}_b &\leftarrow \lceil y_b^k \rceil, \qquad \hat{l}_j \leftarrow l_j \quad \forall j \neq b \\ p &\equiv \{(l, \hat{u})\}, n = \rho(p) \\ q &\equiv \{(\hat{l}, u)\}, n = \rho(q) \end{aligned}$$

  and $R = R\backslash\{n\} \cup \{p\} \cup \{q\}$. goto Step 1.
else if $(\hat{x}, \hat{y})$ integer **then**
  if $z_{\text{NLPR}(l,u)} \leq UB$ **then**
    Update upper bound information: $UB = z_{\text{NLPR}(l,u)}$; $(x^*, y^*) = (\hat{x}, \hat{y})$.
  end if
  for all $r \in R$ **do**
    if $Z^r \geq UB$ **then**
      $R = R\backslash\{r\}$ (Fathom nodes).
    end if
  end for
  Goto Step 1.
end if

Algorithm 1: Nonlinear Branch and Bound Algorithm

The idea is to branch on the variable y_i for which the expected increase in the objective function is the largest. For each integer variable y_i, let the down-pseudocost, e_i^-, and the up-pseudocost, e_i^+, refer to the estimates of the per unit change in the objective function value of the two child nodes obtained by branching down and up on variable y_i. The estimated change in the objective value for the two child nodes may then be estimated as $D_i^- = e_i^- y_i^*$, and $D_i^+ = e_i^+(1 - y_i^*)$ respectively. A score V_i, is then computed which takes into account the improvements over both branches. One way to do this is to compute $V_i = \min\{D_i^-, D_i^+)\}$ for each variable y_i, and then find the branching variable y_i for which the estimate V_i is the maximum.

One way to compute the down and up-pseudocosts is to observe the improvement in the objective value by evaluating the two child nodes for each candidate branching variable. Let Z_i^- and Z_i^+ be the objective values of the two child nodes obtained by branching on y_i. The pseudocosts can then be explicitly computed by calculating

$$e_i^- = \frac{Z_i^- - Z^*}{y_i^*}, \qquad e_i^+ = \frac{Z_i^+ - Z^*}{1 - y_i^*}. \tag{1.4.35}$$

This explicit computation before branching usually turns out to be very expensive computationally. Typically, the way pseudocosts are used is to initialize them to some value and later update them efficiently. Pseudocost initialization is usually done by either using some average observed value, or using objective function coefficients, or by explicit computation. Once a branching variable is selected by using these pseudocosts, the pseudocost estimates for a variable y_i are updated to relect the historical success achieved in branching on that

variable. A simple way to do this is to simply observe the true pseudocosts for a variable after the two child nodes have been evaluated. Using the initial pseudocost, the latest pseudocost, or the average of such observations for each variable are some of the ways for updating pseudocosts. See Linderoth and Savelsbergh [1999] for different strategies dealing with pseudocost initialization and updation.

Also, the reader is referred to Linderoth and Savelsbergh [1999] and Achterberg et al. [2004] for a detailed exposition on branching rules for MILPs. The following node selection rules may also be employed:

1. Depth-first search. This selects the deepest node in the tree from the list of open nodes R. This minimizes the number of nodes that need to be stored. Also, this strategy may be useful to obtain feasible solutions since they are likely to be found deeper in the tree.

2. Best lower bound. In this strategy, the node which currently has the greatest lower bound on the objective is selected for evaluation. This minimizes the number of nodes that need to be evaluated before terminating the tree search.

3. Best estimate. In this approach, an estimate of the value of the best solution obtainable from each open node is calculated and the node corresponding to the best estimate is chosen. The idea is to select a node that may lead to a better integer feasible solution. The best estimate for an open node $n \in R$, E^n, is calculated by using pseudocosts as follows:

$$E^n = Z^n + \sum_i \min(e_i^- y_i^*, e_i^+ (1 - y_i^*)).$$

The reader is referred to Linderoth and Savelsbergh [1999] for an in-depth study of node selection strategies in the branch-and-bound framework for MILPs.

Borchers and Mitchell [1994] developed a branch-and-bound scheme for convex mixed 0-1 nonlinear programs. They use a Sequential Quadratic Programming (SQP) method for solving the continuous relaxation at each node. Instead of solving the continuous relaxation to optimality, they examine the current solution after every iteration of the SQP method. They use a heuristic approach to determine whether the subproblem solved at the node yields a fractional solution or not. They calculate the lower bound for the node without having to solve the node to optimality by solving a Lagrangian dual. This method allows for the possibility of early fathoming of the current node, thus speeding up the algorithm. Borchers and Mitchell [1997] did a computational comparison of the branch-and-bound algorithm with the Outer Approximation algorithm (which will be explained in Section 1.4.4) for convex mixed 0-1 nonlinear programs. They suggested that the failure of Outer Approximation algorithm to converge to the optimal solution may have to do with the inability of the algorithm to generate a sufficiently accurate outer-approximation for the MINLP. Fletcher and Leyffer [1998a] came up with an algorithm for solving a convex Mixed Integer Quadratic Programs (MIQP), which is the problem of minimizing a convex quadratic function subject to linear constraints and bounds on discrete and continuous variables, in a general branch-and-bound framework. The idea was to obtain better lower bounds for the nodes. Their approach helped reduce the number of QP problems to be solved. They also conducted numerical experiments to suggest that the branch-and-bound algorithm works better than other algorithms for the MIQP problem.

1.4.3 Cutting Plane Methods

Cutting planes are constraints that separate a solution of a continuous relaxation of a problem from the set of feasible solutions to the problem. Cutting plane methods have become very useful in solving MILPs. See Cornuejols [2008], and Marchand et al. [1999] for a review of the various cutting planes in mixed-integer linear programming. Cutting planes are sometimes obtained from relaxations of specific polyhedral sets by exploiting the problem structure (see Crowder et al. [1983]). Examples of relaxation based cuts include knapsack covers, flow cover inequalities and clique inequalities (see Padberg [1979], Padberg et al. [1985], Marchand and Wolsey [1999]). There are also general purpose cutting planes that do not make use of the substructures in the problem. Examples of general general purpose cutting planes include Gomory cuts (Gomory [1958]), mixed-integer rounding (MIR) cuts (Marchand and Wolsey [2001]), and lift-and-project cuts (see Balas et al. [1993, 1996]). The cutting planes are typically used in a branch-and-cut framework. The branch-and-cut procedure enhances the branch-and-bound procedure by adding cutting planes to tighten the relaxation that is solved at any node during the tree search.

Some of the earlier work on cutting planes for MINLPs concentrated on deriving cuts for special classes of MINLPs. Granot et al. [1982] came up with a cutting-plane algorithm for pure 0-1 positive polynomial programs, which minimize a linear objective function subject to constraints that are polynomial functions with positive coefficients. They generated linear cover inequalities for the problem that are violated by the current solution obtained by a heuristic procedure. They modified the method by generating deeper cuts by finding the most violated cover inequality. Balas and Mazzola [1984a,b] improved upon the method by finding a stronger dominating

constraint.

Some of the recent work on cutting planes for MINLPs deal with the generalization of cutting planes for MILPs to MINLPs. Cezik and Iyengar [2005] have extended the theory of Gomory cuts for mixed 0-1 conic programs (MICP). The MICP is a special class of MINLP where the objective function is linear and the constraints are convex cones that include linear cones, second-order cones and semidefinite cones. Many classes of nonlinear inequalities can be expressed as conic constraints, leading possibly to nonlinear cutting planes for MINLPs. Stubbs and Mehrohtra [2002] next came up with convex polynomial cuts for MICPs. Atamtürk and Narayanan [2007] have recently extended mixed-integer rounding (MIR) cuts for MICPs. Frangioni and Gentile [2006] have extended the theory of perspective cuts for MIQPs when the objective function has a special structure. Stubbs and Mehrotra [1999] have come up with disjunctive cuts for 0-1 MINLPs in a branch-and-cut framework. Their work is an extension of disjunctive cuts for MILP problems by Balas [1979] and Balas et al. [1993]. They show that such cuts can be generated by solving a single convex program. They further show that the tighter cuts generated by Sherali and Adams [1990, 1994], and Lovász and Schrijver [1991] for the pure 0-1 linear case can also be generated in this more general setting.

There has also been some work done on using general-purpose cutting planes for MILPs in a branch-and-cut framework for solving MINLPs. Akrotirianakis et al. [2001] have used Gomory cuts in a branch-and-cut framework for 0-1 MINLPs. Their computational experiments have shown improvements in terms of the number of nodes of the enumeration tree that need to be investigated. Bonami et al. [2008] have also used Gomory cuts for solving the MILP in their hybrid implementation for solving mixed 0-1 convex MINLPs. These algorithms are based on the algorithm by Quesada

and Grossmann, and will be explained in details in Section 1.4.7.

1.4.4 Outer Approximation

The Outer Approximation (OA) algorithm, proposed by Duran and Grossmann [1986], is based on iteratively solving the NLP subproblem (NLP(y^k)) and an MILP relaxation of (MINLP) to generate a sequence of points (x^k, y^k).

The following assumptions are made regarding the problem being solved:

A.1 X is a non-empty, convex set defined by a system of linear inequalities.

A.2 f and g are convex, and twice continuously differentiable.

A.3 The first order Karush-Kuhn-Tucker (KKT) conditions are necessary to hold at every optimal solution of (NLP(y^k)) and (NLPF(y^k)). Further, a constraint qualification holds at the solution of every nonlinear program (NLP(y^k)), or (NLPF(y^k)). This could, for example, mean that the linear independence constraint qualification (LICQ) holds at that solution, implying that the gradients of the active constraints are linearly independent.

Before describing the algorithm, we first define an outer approximation based MILP which contains linearizations from *an appropriately defined* finite set of integer feasible points in (MINLP). The convexity property of the nonlinear functions f and g_j $(\forall j = 1 \dots m)$ implies linearizations (OA(x^k, y^k)) outer-approximate the feasible region of (MINLP) and under-estimate the objective function. Please see Figures 1.5 and Figures 1.6 for the geometric interpretation of the linearizations.

The OA based master problem using this appropriately defined finite set of integer

feasible points, K^*, is formulated as follows:

$$
\begin{aligned}
Z^K = \min \quad & \eta \\
\text{s.t. } & \eta \geq f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \quad k \in K^*, \\
& g(x^k, y^k) + \nabla g(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0 \quad k \in K^*, \\
& x \in X, y \in Y \cap \mathbb{Z}^p, \eta \in \mathbb{R}^1,
\end{aligned}
\tag{MILP-OA}
$$

where $K^* = \|K_f^* \cup K_i^*\|$, such that $K_f^* = \{k|\ (x^k, y^k) \in \arg\min\ (\text{NLP}(y^k))\ \}$, and $K_i^* = \{k|\ (x^k, y^k) \in \arg\min\ (\text{NLPF}(y^k))\ \}$ when $(\text{NLP}(y^k))$ is infeasible (i.e. $\arg\min\ (\text{NLP}(y^k)) = \emptyset$).

For its derivation, the OA algorithm is based on the following theorem:

Theorem 1.4.1 (Duran and Grossmann [1986], Fletcher and Leyffer [1994], Bonami et al. [2008])**.** *If assumptions **A.1**, **A.2** and **A.3** hold, and K^* is an appropriately defined finite set of integer feasible points, then (MINLP) and the MILP master problem (MILP-OA) have the same optimal value, and that the optimal solution (x^*, y^*) of (MINLP) corresponds to an optimal solution (η^*, x^*, y^*) of (MILP-OA) with $\eta^* = f(x^*, y^*)$.*

We now consider an MILP relaxation $(\text{MP}(\mathcal{K}))$ of (MINLP), built assuming that the solutions of K $(K = |\mathcal{K}|)$ different NLP subproblems are available, where $\mathcal{K} = \{(x^0, y^0), (x^1, y^1), \ldots, (x^{|\mathcal{K}|}, y^{|\mathcal{K}|}\} \subseteq K^*$ is a set of points:

$$
\begin{aligned}
z_{\mathrm{MP}(\mathcal{K})} = \min \quad & \eta \\
\text{s.t. } & \eta \geq f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \quad k = 1 \ldots |\mathcal{K}|, \qquad (\mathrm{MP}(\mathcal{K})) \\
& g(x^k, y^k) + \nabla g(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0 \quad k = 1 \ldots |\mathcal{K}|, \\
& x \in X, y \in Y \cap \mathbb{Z}^p, \eta \in \mathbb{R}^1.
\end{aligned}
$$

Given the assumption of convexity of the functions $f(x, y)$ and $g_j(x, y)$, the following property can be easily established:

Property 1.4.1. *The solution of problem (MP($\mathcal{K}$)), $Z^K = z_{\mathrm{MP}(\mathcal{K})}$, corresponds to a lower bound for the problem (MINLP).*

Since function linearizations are accumulated as iterations proceed, the master problem (MP($\mathcal{K}$)) yields a non-decreasing sequence of lower bounds, $Z^1 \ldots \leq Z^k \leq \ldots \leq Z^K$. Duran and Grossmann [1986] showed that outer approximation provides stronger lower bounds than the Benders Decomposition method (described subsequently in Section 1.4.5).

The OA algorithm begins by solving (NLPR(l, u)). Linearizations about the solution of (NLPR(l, u)) are used to set up the MILP subproblem (MP($\mathcal{K}$)). (MP($\mathcal{K}$)) is then solved to optimality to give an integer solution (x^k, y^k). The NLP subproblem for fixed integer values y^k (NLP(y^k)) is then solved. If (NLP(y^k)) is feasible, linearizations (OA(x^k, y^k)) are added to the master problem. These linearizations cut off the current solution (x^k, y^k). Also, $z_{\mathrm{NLP}(y^k)}$ provides an upper bound to (MINLP).

If (NLP(y^k)) is infeasible, the feasibility subproblem (NLPF(y^k)) is solved and the

linearizations (FC(x^k, y^k)) generated about its solution and added to the master MILP (MP($\mathcal{K}$)). Fletcher and Leyffer [1994] showed that the linearizations (FC(x^k, y^k)) cut off the infeasible assignment (x^k, y^k). This iterative procedure of solving (NLP(y^k)) and (MP($\mathcal{K}$)) and updating the formulation of (MP($\mathcal{K}$)) is continued till the lower and upper bound are within a specified tolerance. Algorithm 2 is a detailed description of the Outer Approximation method.

Initialization: $k := 0$, $UB := \infty$.
Solve NLP relaxation (NLPR(l, u)). Solution is (x^k, y^k).
Add linearizations (OA(x^k, y^k)) about (x^k, y^k) to the MILP (MP($\mathcal{K}$)).
$k := k + 1$.
Step 1: Solve (MP($\mathcal{K}$)).
if (MP($\mathcal{K}$)) is infeasible **then**
  STOP.
else
  Let the integer solution be ($\bar{x}^k, y^k$) and objective value be Z^k.
end if
if $Z^k \geq UB$ **then**
  STOP.
end if
Solve (NLP(y^k)).
if (NLP(y^k)) is feasible **then**
  Let solution be (x^k, y^k) with objective value $z_{\text{NLP}(y^k)}$.
  if $z_{\text{NLP}(y^k)} \leq UB$ **then**
    $UB := z_{\text{NLP}(y^k)}$, $(x^*, y^*) = (x^k, y^k)$.
  end if
  Add constraint (OA(x^k, y^k)) to the master problem (MP($\mathcal{K}$)).
else if (NLP(y^k)) is infeasible **then**
  Solve (NLPF(y^k)). Let solution be (x^k, y^k).
  Add constraint (FC(x^k, y^k)) to the master problem (MP($\mathcal{K}$)).
end if
$k := k + 1$. Goto Step 1.

Algorithm 2: OA Algorithm

The OA method typically requires relatively few cycles or major iterations. One reason for this behavior is given by the following property:

Property 1.4.2. *The OA algorithm converges in one iteration if $f(x, y)$ and $g(x, y)$*

are linear.

This property follows simply from the fact that if $f(x, y)$ and $g(x, y)$ are linear in x and y the MILP master problem (MP($\mathcal{K}$)) is identical to the original problem (MINLP). The following theorem gives the results regarding the finite convergence of the algorithm:

Theorem 1.4.2. *If assumptions **A.1**, **A.2** and **A.3** hold, and the set Y is finite, then the OA algorithm terminates in a finite number of steps at an optimal solution of MINLP or with an indication that the problem (MINLP) is infeasible.*

It is also important to note that the MILP master problem need not be solved to optimality. In fact, given the upper bound UB^k and a tolerance ϵ, it is sufficient to generate a new (x^k, y^k) by solving:

$$
\begin{aligned}
Z^K = \min \quad & \eta \\
\text{s.t. } & \eta \geq f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \quad k = 1 \ldots K \qquad \text{(RM-OAF)} \\
& g(x^k, y^k) + \nabla g(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0 \quad k = 1 \ldots K \\
& \eta \leq UB^k - \epsilon \quad k = 1 \ldots K \\
& x \in X, y \in Y \cap \mathbb{Z}^p, \eta \in \mathbb{R}^1.
\end{aligned}
$$

When nonlinear functions in the problem are not adequately represented by linear approximations in the MILP, the OA algorithm may perform badly. Fletcher and Leyffer [1994] suggested the use of curvature information into the master program. This was done by including a second order Lagrangian term into the objective function

of the MILP master problem. The resulting formulation is of the form:

$$
\begin{aligned}
Z^K = \min \quad & \eta + \frac{1}{2} \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix}^T \nabla^2 \mathcal{L}(x^k, y^k) \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \\
\text{s.t. } & \eta \geq f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \quad k = 1 \ldots K \qquad \text{(M-MIQP)} \\
& g(x^k, y^k) + \nabla g(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0 \quad k = 1 \ldots K \\
& \eta \leq UB^k - \epsilon \quad k = 1 \ldots K \\
& x \in X, y \in Y \cap \mathbb{Z}^p, \eta \in \mathbb{R}^1.
\end{aligned}
$$

In the definition of (M-MIQP), the function

$$\mathcal{L}(x^k, y^k) = f(x^k, y^k) + \lambda^T g(x^k, y^k)$$

is the usual Lagrangian function obtained from the NLP subproblem (NLP(y^k)), and $\nabla^2 \mathcal{L}(x^k, y^k)$ is the Hessian of this Lagrangian function. Note that Z^K does not provide valid lower bounds in this case. However, the linear part of the objective, η, may be used to obtain lower bounds. The authors have shown that the original OA algorithm may require many more iterations to converge than when the master problem (M-MIQP) is used. This, however comes at the price of having to solve an MIQP instead of an MILP. Fletcher and Leyffer [1994] also extend the algorithm of Duran and Grossmann [1986] to handle problems where nonlinearities are also associated with the integer variables, thus making OA algorithm applicable to a wider range of MINLPs. Their work also deals with methods for handling infeasibility of

the NLP problems. They also investigate the worst case performance of the OA algorithm.

Outer Approximation has been implemented as a commercial solver DICOPT (Kocis and Grossmann [1987]). A limitation of the outer approximation method lies in its inability to handle nonconvexity in the problem. Heuristics to remedy this have come up in the literature, notably ones by Kocis and Grossmann [1989] and Viswanathan and Grossmann [1990]. These methods deal with relaxation of the nonlinear equality constraints and with reducing the effects of nonconvexity by penalizing constraints.

Another noteworthy work in this area is of Michelon and Maculan [1991], who came up with a Lagrangian decomposition method for integer nonlinear programs with linear constraints. Solving the dual Lagrangian relaxation, they obtain at each iteration, the solution of a nonlinear programming problem with continuous variables and an integer linear programming problem. They proposed two algorithms aimed at reducing the duality gap. Their algorithm employs linear supporting hyperplanes on the objective function that cut off any integer assignments that have already been visited by the algorithm. This makes it possible to interpret the master program in the Lagrangian decomposition as an Outer Approximation master program.

1.4.5 Generalized Benders Decomposition

Originally, Benders Decomposition was introduced by Benders [1962] for a problem that is linear in the "easy" variables, and nonlinear in the "complicating" variables. Employing duality theory for nonlinear programs, Geoffrion [1972] came up with a Generalized Benders Decomposition (GBD) method for solving a larger class of problems.

The GBD method (also see Flippo and Kan [1993]) is very similar to the OA method. The difference lies in the definition of the MILP master problem. Instead of using supporting hyperplanes (as in Outer Approximation), Benders Decomposition makes use of the duality theory to derive an equivalent formulation (RM-GBD). The advantage of this formulation is that it does not contain the continuous variables $\mathbf{x}$ (instead, it contains only one continuous variable η) and that only one constraint is added every iteration. The disadvantage, however, is that the cuts derived from Benders decomposition are weaker than the corresponding linearizations from Outer Approximation.

Let (x^k, y^k) be the solution of (NLP(y^k)) for a given integer assignment y^k. The linearizations about this point are given by (OA(x^k, y^k)). The Karush-Kuhn-Tucker (KKT) conditions at the solution of (NLP(y^k)) are given by

$$\begin{aligned} \nabla_x f(x^k, y^k) + (\mu^k)^T \nabla_x g(x^k, y^k) &= 0, \\ (\mu^k)^T g(x^k, y^k) &= 0, \\ \mu^k &\geq 0, \end{aligned}$$

where μ^k are the optimal Lagrange multipliers, ∇_x and ∇_y refer to the gradients of functions (f or g) with respect to only the x, and y variables respectively.

Multiplying the optimal multipliers μ^k with linearizations of g, and adding them with the linearization of f gives us the following inequality:

$$\begin{aligned} \eta \geq & f(x^k, y^k) + \nabla_x f(x^k, y^k)^T (x - x^k) + \nabla_y f(x^k, y^k)^T (y - y^k) + \\ & (\mu^k)^T (g(x^k, y^k) + \nabla_x g(x^k, y^k)^T (x - x^k) + \nabla_y g(x^k, y^k)^T (y - y^k)). \end{aligned} \tag{1.4.36}$$

By combining KKT conditions with (1.4.36), the continuous variables x are eliminated, and this leads to the inequality:

$$\eta \geq f(x^k, y^k) + (\nabla_y f(x^k, y^k) + (\mu^k)^T \nabla_y g(x^k, y^k))^T (y - y^k), \qquad (\text{BC}(x^k, y^k))$$

which is the Lagrangian cut projected in the y-space. For the case when there is no feasible solution to (NLP(y^k)), then if the point x^k is obtained from the feasibility subproblem (NLPF(y^k)), the following feasibility cut projected in y can be obtained using a similar procedure,

$$(\lambda^k)^T [g(x^k, y^k) + \nabla_y g(x^k, y^k)^T (y - y^k)] \leq 0. \qquad (\text{FCY}(x^k, y^k))$$

In this way, the master problem (RM-GBD) may then be formulated as:

$$\begin{aligned} Z^K = \min \quad & \eta \\ \text{s.t. } & \eta \geq f(x^k, y^k) + \nabla_y f(x^k, y^k)^T (y - y^k) + \\ & (\mu^k)^T [g(x^k, y^k) + \nabla_y g(x^k, y^k)^T (y - y^k)] \quad \forall k \in \text{KFS}, \qquad (\text{RM-GBD}) \\ & (\lambda^k)^T [g(x^k, y^k) + \nabla_y g(x^k, y^k)^T (y - y^k)] \leq 0 \quad \forall k \in \text{KIS}, \\ & y \in Y \cap \mathbb{Z}^p, \eta \in \mathbb{R}^1. \end{aligned}$$

where KFS is the index set of feasible subproblems and KIS is the index set of infeasible subproblems whose solution is given by (NLPF(y^k)). The pseudo-code for the GBD Algorithm is provided in Algorithm 3.

Balas [1969] was the first to extend the idea of Benders decomposition to the case of quadratic objective function. Later, Flippo and Kan [1990] gave a correct

Initialization: $k := 0$, $UB := \infty$.
Solve NLP relaxation (NLPR(l, u)). Solution is (x^k, y^k).
Add linearizations (BC(x^k, y^k)) about (x^k, y^k) to the GBD Master MILP (RM-GBD). $k := k + 1$.
Step 1: Solve GBD MILP (RM-GBD).
if (RM-GBD) infeasible **then**
  STOP.
else
  Let the integer solution be $(\bar{x}^k, y^k)$ and objective value Z^k.
end if
if $Z^k \geq UB$ **then**
  STOP.
end if
Solve (NLP(y^k)).
if (NLP(y^k)) is feasible **then**
  Let solution be (x^k, y^k) with objective value $z_{\text{NLP}(y^k)}$.
  if $z_{\text{NLP}(y^k)} \leq UB$ **then**
    $UB := z_{\text{NLP}(y^k)}$, $(x^*, y^*) = (x^k, y^k)$.
  end if
  Add constraint (BC(x^k, y^k)) to the master problem (RM-GBD).
else if (NLP(y^k)) is infeasible **then**
  Solve (NLPF(y^k)). Let solution be (x^k, y^k).
  Add constraint (FCY(x^k, y^k)) to the master problem (RM-GBD)
end if
$k := k + 1$. Goto Step 1.

Algorithm 3: GBD Algorithm

interpretation of Benders Decomposition for the MIQP case, improving upon the previous work in this regard. Sahinidis and Grossmann [1991] have derived some convergence properties for the GBD method.

An important property that shows the relation between the Generalized Benders Decomposition and Outer Approximation methods is given as:

Property 1.4.3. *Given the same set of K subproblems, the lower bound predicted by the relaxed master problem (MP($\mathcal{K}$)) is greater than or equal to the one predicted by the relaxed master problem (RM-GBD).*

The above property follows from the fact that the Lagrangian (BC(x^k, y^k)) and feasibility cuts (FCY(x^k, y^k)) are weaker than the outer-approximation constraints (OA(x^k, y^k)). Given the fact that the lower bound of GBD are weaker, this method commonly requires a large number of cycles or major iterations. As the number of 0-1 variables increase, this difference becomes more pronounced. This is to be expected since only one cut is generated per iteration. Therefore, user-supplied constraints must often be added to the master problem to strengthen the bounds. Also, it is sometimes possible to generate multiple cuts from the solution of an NLP subproblem in order to strengthen the lower bound. As for the OA algorithm, the trade-off is that while it generally predicts stronger lower bounds than GBD, the computational cost for solving the master problem is greater since the number of constraints added per iteration is equal to the number of nonlinear constraints plus the nonlinear objective.

1.4.6 Extended Cutting Plane Method

Westerlund and Pettersson [1995] proposed the Extended Cutting Plane (ECP) method for convex MINLPs. The method can be viewed as an extension of the cutting plane

method of Kelley [1960], which was developed for solving convex NLPs. The method does not rely on the use of NLP subproblems. The algorithm is based on iteratively solving an MILP (M-ECP), and adding a linearization of the most violated constraint at the optimal solution (x^k, y^k) of (M-ECP) at every iteration.

The MILP master problem (M-ECP) at iteration K is defined as:

$$
\begin{aligned}
Z^K = \min \quad & \eta \\
\text{s.t. } \eta \geq\ & f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \quad k = 1 \dots K \qquad \text{(M-ECP)} \\
& g_j(x^k, y^k) + \nabla g_j(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0 \quad k = 1 \dots K, \quad j \in J^k \\
& x \in X, y \in Y \cap \mathbb{Z}^p, \eta \in \mathbb{R}^1,
\end{aligned}
$$

where $J^k = \{\hat{j} \in \arg\{\max_{j=1\dots m} g_j(x^k, y^k)\}\}$ corresponds to the index set of the violated constraints at iteration k.

The convexity of the nonlinear functions f and g_j is exploited by replacing them with supporting hyperplanes ($\text{OA}(x^k, y^k)$) that are derived at a point (x^k, y^k) by evaluating gradients instead of solving an NLP subproblem ($\text{NLP}(y^k)$). The procedure of adding a linearization to the MILP at every iteration and resolving the MILP to optimality generates a non-decreasing sequence $Z^1, Z^2, \dots, Z^K$ of lower bounds. Convergence is achieved when the maximum constraint violation at any iteration is within a specified tolerance. It is also possible to add to (M-ECP), linearizations of all the violated constraints.

It is to be noted that the ECP method may require a large number of iterations since only one linearization is added at every iteration, and because NLP subproblems

are not solved about an integer assignment y^k. However, when the objective and constraint functions are linear, the algorithm trivially converges in one iteration.

Westerlund et al. [1998], Westerlund and Pörn [2002] extended this method to deal with pseudo-convex MINLPs. Recall that a function $f : S \to \mathbb{R}$ defined on a convex subset $S \subseteq \mathbb{R}^n$ is defined to be pseudo-convex, if for any $x, y \in S$, we have

$$\nabla f(x)^T(y - x) \geq 0 \Rightarrow f(y) \geq f(x).$$

1.4.7 LP/NLP-Based Branch-and-Bound

The LP/NLP-based Branch-and-Bound algorithm (LP/NLP-BB) was proposed by Quesada and Grossmann [1992] to solve (MINLP). LP/NLP-BB is a clever extension of Outer Approximation, that solves an alternating sequence of NLP subproblems and MILP master problems. The NLP subproblem (NLP(y^k)) is obtained by fixing the integer variables at y^k, and the MILP master problem accumulates linearizations (outer approximations of the feasible set, and underestimators of the objective function) from the solution of (NLP(y^k)).

The method starts by solving the NLP relaxation (NLPR(l, u)), and use the solution of (NLPR(l, u)) to set up the MILP (MP($\mathcal{K}$)), defined in Section 1.4.4. The basic idea is to perform a branch-and-bound enumeration of (MP($\mathcal{K}$)) in the search for integer solutions. LP/NLP-BB avoids solving multiple MILPs by interrupting the MILP tree search whenever an integer solution $(\hat{x}, y^k)$ is found to solve the NLP subproblem (NLP(y^k)). The outer approximations from (NLP(y^k)) are then used to update the representation of the MILP master in the open nodes of the tree, and the MILP tree search continues. Thus, instead of solving a sequence of MILPs, as is the

case in the Outer Approximation method, only a single MILP tree search is required.

The key feature in the algorithm is the dynamic generation of the linear approximations that are derived at integer solutions in the tree. If (NLP(y^k)) is feasible, then $z_{\text{NLP}(y^k)}$ provides an upper bound to (MINLP). However, it may turn out that (NLP(y^k)) is infeasible. In this case, one needs to solve the feasibility problem (NLPF(y^k)) to obtain linearizations that exclude the current infeasible point y^k. Some NLP solvers detect the infeasibility of (NLP(y^k)) and proceed to solve (NLPF(y^k)). One way to avoid solving the feasibility problem when the discrete variables in (MINLP) are 0-1, is to introduce the following integer cut whose objective is to make infeasible the choice of the previous 0-1 values generated at the K previous integral nodes for which (NLP(y^k)) is infeasible. The integer cut is as follows:

$$\sum_{i \in B^k} y_i - \sum_{i \in N^k} y_i \leq |B^k| - 1 \quad k = 1..K$$

where $B^k = \{i | y_i^k = 1\}$, $N^k = \{i | y_i^k = 0\}$, $k = 1..K$. However, this cut becomes very weak as the dimensionality of the 0-1 variables increases.

From the solution to (NLP(y^k)) or (NLPF(y^k)), we can derive outer-approximation linearizations (OA(x^k, y^k)) for (MINLP). These inequalities are used to create a master MILP. Given a set of points $\mathcal{K} = \{(x^0, y^0), (x^1, y^1), \ldots, (x^{|\mathcal{K}|}, y^{|\mathcal{K}|})\}$, an outer approximation master MILP (MP($\mathcal{K}$)) for the original MINLP is defined in Section 1.4.4. LP/NLP-BB relies on solving the continuous relaxation to (MP($\mathcal{K}$)) and enforcing integrality of the y variables by branching. We label this problem as CMP($\mathcal{K}, l, u$).

$$\begin{aligned} &\text{minimize} \quad \eta \\ &\quad \text{s.t.} \ (\eta, x, y) \in \mathcal{OA}(\mathcal{K}), \qquad (\text{CMP}(\mathcal{K}, l, u)) \end{aligned}$$

$$x \in X, y \in Y \cap [l, u], \eta \in \mathbb{R}^1,$$

where $\mathcal{OA}(\mathcal{K})$ is the set of points satisfying the outer approximations at the set of points $\mathcal{K}$.

$$\mathcal{OA}(\mathcal{K}) = \{(\eta, x, y) \in \mathbb{R}^{1+n+p} \mid (x, y, \eta) \text{ satisfy } \mathrm{OA}(x^k, y^k)\ \forall (x^k, y^k) \in \mathcal{K}\}.$$

If $\mathcal{K}$ in (MP($\mathcal{K}$)) contains an appropriately-defined finite set of points (these points will be identified by the LP/NLP-BB Algorithm 4), then under mild conditions, $z_{\text{MINLP}} = z_{\text{MP}(\mathcal{K})}$ (Fletcher and Leyffer [1994], Bonami et al. [2008]).

The main algorithm underlying our work in Chapter 2, LP/NLP-BB, is formally stated in pseudo-code form in Algorithm 4. Let (y^l, y^u) refer to the original bounds on y variables in (MINLP). We characterize a node $n \equiv (l, u)$ of the branch-and-bound search tree by the bounds $\{(l, u)\}$, where l is the vector of lower bounds and u is the vector of upper bounds enforced on the integer variables y. The index k is used to track a node n at which the relaxed master problem (CMP($\mathcal{K}, l, u$)) is solved. Let η^k refer to the lower bound for the subtree rooted at node $\{n\}$. For the sake of brevity in the pseudo-code, we refer to a node as $n \equiv (l, u, \eta^k)$. We denote the parent of the node n by $\rho(n)$. The index bk refers to the value of k the last time the **(branch)** portion of the algorithm was executed. We defer discussion of the **(cut)** portion of the LP/NLP-BB algorithm until Section 2.3 in Chapter 2.

An important difference between the LP/NLP-BB algorithm and typical branch-and-bound algorithms is that after finding an integer feasible point in LP/NLP-BB, the corresponding node is re-solved. This modification is necessary because the integer point is no longer feasible in the master problem after adding the linearizations about

Solve NLPR(y^l, y^u) **(initialize)**
Let (x^0, y^0) be its solution and η^0 be the objective value.
if NLPR(y^l, y^u) is infeasible **then**
  Stop. MINLP is infeasible
else
  $\mathcal{K} \leftarrow \{(x^0, y^0)\}, \mathcal{L} \leftarrow \{(y^l, y^u, \eta^0)\}, UB \leftarrow \infty, k \leftarrow 0, bk \leftarrow 0$
end if
while $\mathcal{L} \neq \emptyset$ **do**
  Select and remove node (l, u, η^k) from $\mathcal{L}$ **(select)**
  $k \leftarrow k + 1$
  Solve CMP$(\mathcal{K}, l, u)$ and let $(\eta^k, \hat{x}, y^k)$ be its solution. **(evaluate)**
  if CMP$(\mathcal{K}, l, u)$ is infeasible **OR** $\eta^k \geq UB$ **then**
    Fathom node (l, u, η^k). **Goto (select)**.
  end if
  if $y^k \in \mathbb{Z}^p$ **then**
    Solve NLP(y^k). **(update master)**
    if NLP(y^k) is feasible **then**
      Let (x^k, y^k) be solution to (NLP(y^k)).
      $UB \leftarrow \min\{UB, z_{\text{NLP}(y^k)}\}$.
      Remove all nodes in $\mathcal{L}$ whose parent objective value $\eta \geq UB$. **(fathom)**
    else
      Let (x^k, y^k) be solution to NLPF(y^k)
    end if
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{(x^k, y^k)\}$. **Goto (evaluate).**
  else if Do additional linearizations **then**
    See Algorithm 5 **(cut)**
    Goto (evaluate)
  else
    Select b such that $y_b^k \notin \mathbb{Z}$. $bk \leftarrow k$. **(branch)**
    $\hat{u}_b \leftarrow \lfloor y_b^k \rfloor, \quad \hat{u}_j \leftarrow u_j \quad \forall j \neq b.$
    $\hat{l}_b \leftarrow \lceil y_b^k \rceil, \quad \hat{l}_j \leftarrow l_j \quad \forall j \neq b.$
    $p \equiv \{(l, \hat{u})\}, \quad n = \rho(p).$
    $q \equiv \{(\hat{l}, u)\}, \quad n = \rho(q).$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{(l, \hat{u}, \hat{\eta}^k)\} \cup \{(\hat{l}, u, \hat{\eta}^k)\}.$
  end if
end while

Algorithm 4: LP/NLP-BB algorithm.

that point.

By keeping and solving only one MILP (MP($\mathcal{K}$)), the LP/NLP-BB method commonly reduces, quite significantly, the total number of nodes that are enumerated. Leyffer [1993] has reported significant savings with this method. The trade-off, however is that the number of NLP subproblems may increase. Computational experience however, indicates that the number of NLP subproblems remain unchanged. This methodology is particularly suited for MINLP models where the integrality in the problem is the bottleneck for the solution procedure and we would like to reduce the number of MILPs solved in the algorithm. The algorithm offers the flexibility of adding different types of linear approximations with which it is still possible to obtain a tight representation of the feasible region, but without greatly increasing the size of the LP subproblems. We exploit this property to derive more linearizations, and this is explained in details in Section 2.3 of Chapter 2. Finally, one notes that the algorithm is finite if the possible number of different configurations is finite. This is because the constraints that are added cut off that particular integer configuration, and therefore the algorithm does not cycle.

Leyffer [2001] came up with an integrated approach for solving MINLPs, combining the use of NLP and MILP software written by him. The algorithm uses SQP as a nonlinear programming solution method. The algorithm seeks to avoid solving the NLP problem at a node to convergence. Instead, branching is allowed after each iteration of the SQP method. His work improves upon the idea of Borchers and Mitchell [1994] in a number of ways. He shows that the use of Lagrangian dual (as in the work by Borchers and Mitchell [1994]) for calculating the lower bound at the node may be avoided, if the linearizations in the SQP solver are treated as cutting planes.

1.4.8 The Hybrid Algorithm of Bonami et al. [2008]

In a recent paper, Bonami et al. [2008] have come up with an algorithmic framework for solving convex MINLPs based on open-source software components. Their framework uses outer approximations and NLP relaxations to compute the lower bounds and (NLP(y^k)) to get the upper bounds in a flexible branch-and-cut scheme. When only NLP relaxations (NLPR(l, u)) are used to compute the lower bounds, the algorithm becomes a classical branch-and-bound algorithm. When outer approximation MILP and NLP subproblems at fixed y are iteratively solved at the root node, the algorithm becomes a classical OA algorithm. Their hybrid algorithm can be suitably tuned to go from one end (BB algorithm) to the other end (OA algorithm), or anything in between depending on parameter settings. Their hybrid algorithm follows the principle of the LP/NLP based branch-and-bound algorithm, since (NLP(y^k)) is solved only when the master problem gives an integer feasible solution. The authors note that this is the least that can be done to ensure that the algorithm converges. However, it is entirely possible to solve NLP subproblems at additional nodes in order to reduce the size of the branch-and-bound tree. This is being done in two ways in the proposed framework: by solving the NLP relaxation at additional nodes of the tree and by performing local enumerations at nodes of the tree, specifically at the root node, to get a good upper bound.

The implementation has been done using open-source components from COIN library. In particular it uses Cbc, Cgl, Clp for the MILP framework and IPOPT, an interior-point NLP solver (Wächter and Biegler [2006]). Using Cbc as the underlying framework for the hybrid procedure has helped to take advantage of the advanced MILP techniques already implemented in or utilized by Cbc. Cbc uses several families

of cuts from Cgl to tighten the OA relaxations. The authors have tried using mixed-integer Gomory cuts, probing cuts, mixed-integer rounding cuts and cover cuts with some success. In Chapter 2, we will compare this solver to a solver that we will propose for solving convex MINLPs.

1.5 Thesis Outline

The remainder of the thesis is organized into three parts. In Chapter 2, we introduce an algorithmic framework based on the LP/NLP algorithm for solving convex MINLPs. We implement a linearization based algorithm in a branch-and-cut framework using existing software components. We make use of the MILP framework's advanced features, like heuristics, cutting planes, branching and node selection schemes, for solving the problem. We present detailed computational results which helps us in deciding which features work best for solving MINLPs. We offer new suggestions for generating and managing linearizations that are shown to be efficient on a wide range of MINLP instances. We do extensive computational experiments, testing the various features in the solver and compare it with existing MINLP solvers.

In Chapter 3, we investigate different heuristic schemes for finding feasible solutions to MINLP based on the principle of the feasibility pump of Fischetti et al. [2005]. We first explain an approach that is based on nonlinear programming techniques and rounding schemes. We study different rounding strategies and propose new rounding schemes. We then suggest ways to improve the feasibility pump scheme of Bonami et al. [2006] which alternates between solving a MILP and an NLP for obtaining integer feasible solutions. We then explain the third approach, that integrates the NLP solve within the tree search for a MILP. This approach is akin to the LP/NLP

approach and has a number of advantages. We pinpoint some difficulties involved with this approach as well. We finally present detailed computational results, comparing the three algorithms and show the effectiveness of our approach for getting good feasible solutions.

In Chapter 4, we study an important class of programs called Mixed Variable Programs (MVP) and investigate means to model them as MINLPs, thereby increasing the domain of applications for which mixed-integer nonlinear programming techniques can be useful. We illustrate our approach on a thermal insulation problem. We first explain the MVP model for the thermal insulation model and highlight the various difficulties involved in modeling it as a MINLP. The difficulties include the use of categorical variables, discontinuities, and functions for which gradients are difficult to compute. Using integer and nonlinear modeling techniques, we next come up with three different MINLP models with varying degrees of smoothness. We explain the pros and cons of our approach. We then use MINLP solution techniques to solve the problem and present detailed computational results which justifies our approach.

Finally, we conclude with the main contributions of this thesis and provide suggestions for future research in this field.

Chapter 2

FilMINT: An Outer Approximation-Based Solver for Mixed Integer Nonlinear Programs

In this chapter, we combine advancements in the field of mixed integer linear programming and nonlinear programming to build a framework for solving MINLPs. The end result of this work is a new solver for convex MINLPs. The solver is based on the LP/NLP algorithm of Quesada and Grossmann [Quesada and Grossmann, 1992] that was described in Section 1.4.7 of Chapter 1. Our new solver, FilMINT, combines the MINTO branch-and-cut framework for MILP with filterSQP used to solve the nonlinear programs that arise as subproblems in the algorithm. The MINTO framework allows us to easily employ cutting planes, primal heuristics, and other well-known MILP enhancements for MINLPs. We present detailed computational experiments that show the benefit of such advanced MILP techniques. We offer new techniques for generating and managing linearizations that are shown to be efficient on a wide range

of MINLPs. Comparisons to existing MINLP solvers are presented, that highlight the effectiveness of our new solver FilMINT.

2.1 Introduction

We consider the development of a new efficient solver for mixed integer nonlinear programs. Recall from Section 1.2 that MINLP can be expressed as

$$\begin{aligned} z_{\text{MINLP}} = \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & g(x,y) \leq 0, \\ & x \in X, y \in Y \cap \mathbb{Z}^p, \end{aligned} \tag{MINLP}$$

where $f : \mathbb{R}^{n\times p} \to \mathbb{R}$ and $g : \mathbb{R}^{n\times p} \to \mathbb{R}^m$ are twice continuously differentiable functions, x and y are continuous and discrete variables, respectively, X is a polyhedral subset of $\mathbb{R}^n$, and Y is a bounded polyhedral subset of $\mathbb{Z}^p$. In this work, we focus on the case where the functions f and g_j are convex, so that by relaxing the restriction $y \in \mathbb{Z}^p$, a convex program is formed. The algorithms that we describe may be applied as a heuristic in the case that one or more of the functions are nonconvex.

The aim of the work described in this chapter is to develop an algorithmic scheme and a solver that is capable of solving MINLPs at a cost that is a small multiple of the cost of a comparable mixed-integer linear program. In our view, the algorithm most likely to achieve this goal is LP/NLP-based branch-and-bound (LP/NLP-BB). As described in Section 1.4.7, this method is similar to outer approximation; but instead of solving an alternating sequence of MILP master problems and nonlinear programs, it interrupts the solution of the MILP master whenever a new integer assignment is

found, and solves an NLP subproblem. The solution of this subproblem provides new outer approximations that are added to the master MILP, and the solution of the updated MILP master continues. Thus, only a single branch-and-cut tree is created and searched.

Our solver exploits recent advances in nonlinear programming and mixed-integer linear programming to develop an efficient implementation of LP/NLP-BB. Our work is motivated by the observation of Leyffer [1993] that a simplistic implementation of this algorithm often outperforms nonlinear branch-and-bound and outer approximation by an order of magnitude. Despite this clear advantage, however, there had been no implementation of LP/NLP-BB until the recent independent work by Bonami et al. [2008] and this work. Our implementation, called FilMINT, is built on top of the mixed-integer programming solver MINTO (Nemhauser et al. [1994]). Through MINTO's branch-and-cut framework, we have been able to exploit a range of modern MILP features, such as enhanced branching and node selection rules, primal heuristics, preprocessing, and cut generation routines. To solve the NLP subproblems, we use filterSQP (Fletcher et al. [2002]), an active set solver with warm-starting capabilities that can take advantage of good initial primal and dual iterates.

Recently, Bonami et al. [2008] have also developed a solver for MINLPs called Bonmin. This solver was introduced in Section 1.4.8 of Chapter 1. While the two solvers share many of the same characteristics, our work differs from Bonmin in a number of significant ways. First, FilMINT differs from Bonmin in the way in which linearizations are added, and how these linearizations are managed. We derive additional linearizations at fractional integer points to strengthen the outer approximation, and we exploit cut management features in MINTO that allow us to keep a much smaller set of linearizations. Second, FilMINT uses an active-set solver for the

NLP subproblems, which allows us to exploit warm-starting techniques that are not readily available for the interior-point code IPOPT (Wächter and Biegler [2006]) that is used in Bonmin. We also note that MINTO's suite of advanced integer programming techniques is different from that of CBC (Forrest [2004]), which is the MILP framework on which Bonmin is based. Finally, Bonmin is a hybrid between a branch-and-bound solver based on nonlinear relaxations and one based on polyhedral outer approximations, whereas FilMINT relies solely on linear underestimators, which we believe to be more efficient in practice, especially for convex MINLP. A comparison of performance of the two solvers is given in Section 2.5.

This chapter is organized as follows: In the remainder of this section, we describe the implementation of FilMINT within MINTO's branch-and-cut framework, and outline the computational setup for our experiments. In Section 2.2, we report a set of careful experiments that show the effect of modern MILP techniques on an LP/NLP-based algorithm. In Section 2.3, we consider several ways to generate additional linearizations for the algorithm and how to use these mechanisms effectively via a cut management strategy. Section 2.4 gives a comparison of FilMINT to other MINLP solvers, and conclusions about this portion of the thesis are offered in Section 2.5.

2.1.1 Implementation within the MINTO Framework

FilMINT is built on top of MINTO's branch-and-cut framework, using filterSQP to solve the NLP subproblems. MINTO provides *user application functions* through which the user can implement a customized branch-and-cut algorithm, and FilMINT is written entirely within these user application functions. No changes are necessary to the core MINTO library in order to implement the LP/NLP-BB algorithm. MINTO can be used with any linear programming solver that has the capability to modify and

resolve linear programs (LP) and interpret their solutions. In our experiments, we use the `Clp` linear programming solver that is called through its `OsiSolverInterface`. Both `Clp` and the `OsiSolverInterface` are open-source tools available from COIN-OR (`http://www.coin-or.org`).

FilMINT obtains problem information from AMPL's ASL interface (Fourer et al. [1993], Gay [1997]). ASL provides the user with gradient and Hessian information for nonlinear functions, which are required by the NLP solver and are used to compute the linearizations (OA(x^k, y^k)) required for LP/NLP-BB. FilMINT's NLP solver, filterSQP, is a sequential quadratic programming (SQP) method that employs a filter to promote global convergence from remote starting points. A significant advantage of using an active-set SQP method in this context is that the method can readily take advantage of good starting points. We use as the starting point the solution of corresponding the LP node, namely, $(\eta^k, \hat{x}, y^k)$. Another advantage of using filterSQP for implementing (LP/NLP-BB) is that filterSQP contains an automatic restoration phase that enables it to detect infeasible subproblems reliably and efficiently. The user need not create and solve the feasibility problem (NLPF(y^k)) explicitly. Instead, filterSQP returns the solution of (NLPF(y^k)) automatically.

The user application functions of MINTO that have been used by FilMINT are `appl_mps`, `appl_feasible`, `appl_primal`, and `appl_constraints`. A brief description of FilMINT's use of these functions is stated next.

- `appl_mps`: The MINLP instance is read. This corresponds to step **(initialize)** in Algorithm 4. The linear constraints in the problem, if any, are also added within this function.

- `appl_feasible`: This user application function allows the user to verify that a

solution to the active formulation satisfying the integrality conditions is feasible. When we generate an integral solution y^k for the master problem, the NLP subproblem (NLP(y^k)) is solved, and its solution provides an upper bound and a new set of outer approximation cuts.

- `appl_constraints`. This function allows the user to generate violated constraints. The solution of (NLP(y^k)) or (NLPF(y^k)) in `appl_feasible` generates new linearizations. These are stored and added to the master problem (CMP($\mathcal{K}, l, u$)) by this method. This function is also used to implement NLP solves at fractional LP solutions, an enhancement that will be explained in Section 2.3.

- `appl_primal`. This function allows the user to communicate a new upper bound and primal solution to MINTO, if the solve of (NLP(y^k)) resulted in an improved feasible solution to (MINLP).

Figure 2.1 shows a flowchart of the LP/NLP-BB algorithm and the MINTO application functions used by FilMINT. We note that, for the sake of simplicity, the figure does not show all the details of the algorithm.

2.1.2 Computational Setup and Preliminary Implementation

In this section, we describe the computational setup and provide an initial comparison of LP/NLP-BB to a standard MINLP branch-and-bound solver. Our aim is to explore the usefulness of the wide range of MILP techniques that MINTO offers in the context of solving MINLP problems. We believe that this study is of interest beyond the scope of the specific LP/NLP-BB algorithm since it may provide an indication of which MILP techniques are likely to be efficient in other methods for solving MINLPs,

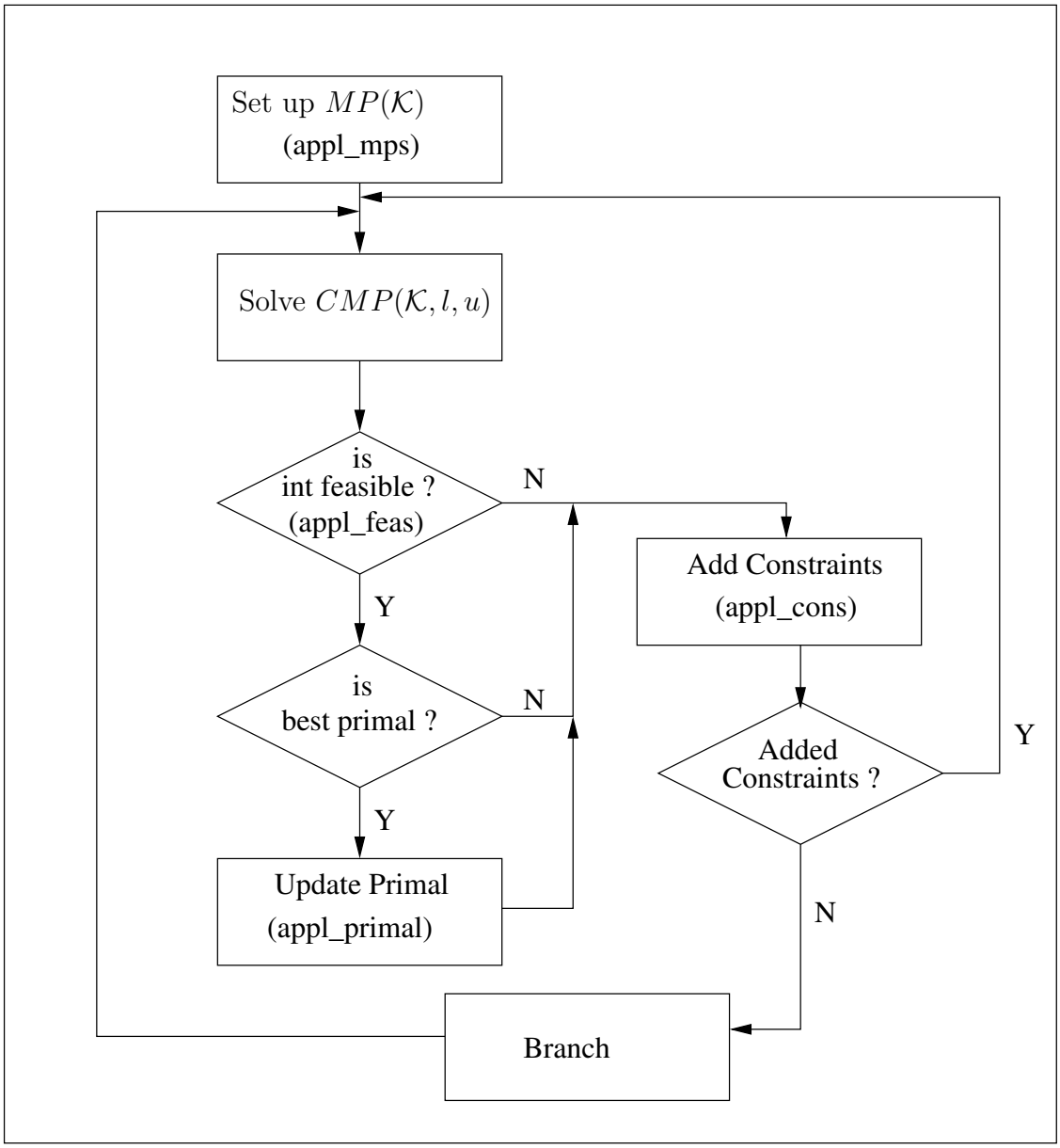

Figure 2.1: LP/NLP-BB implementation within MINTO.

such as branch-and-bound using the relaxation (NLPR(l, u)). We carry out a set of carefully constructed computational experiments to discover the salient features of a MILP solver that have the biggest impact on solving MINLPs.

The test problems have been collected from the GAMS collection of MINLP problems (Bussieck et al. [2003]), the MacMINLP collection of test problems (Leyffer [2003]), and the collection on the website of IBM-CMU research group (Sawaya et al. [2006]). Since FilMINT accepts only AMPL as input, all GAMS models were converted into AMPL format. The test suite comprises 219 convex problems covering a wide range of applications.

The experiments have been run on a cluster of (identical) computers at Lehigh University. The cluster consists of 120 nodes of 64-bit AMD Opteron microprocessors. Each of the nodes has a CPU clockspeed of 1.8 GHz, and 2 GB RAM and runs on Fedora Core 2 operating system. We note that the current release of FilMINT is not parallel, so that only one node of the cluster is used to solve a particular instance. Since all the nodes are of the same architecture and capacity, we can use the results of test problems running on different nodes to make valid comparisons. All of the codes we tested were compiled by using the GNU suite of C, C++, and FORTRAN compilers.

The test suite of convex instances have been categorized as easy, moderate, or difficult, based on the time taken using MINLP-BB (Leyffer [1998]), a branch-and-bound solver based on the relaxation (NLPR(l, u)), to solve each instance. The easy convex instances take less than one minute to solve. Moderate convex instances take between one minute to one hour to solve. The difficult convex instances are not solved in one hour. There are 97 easy, 37 moderate, and 85 difficult instances in the testsuite, the names and characteristics of these instances can be found in the

Appendix in Tables A.1–A.3.

Experiments have been conducted by running the test problems using FilMINT (with various features set on or off) for a time limit of four hours. We create performance profiles (see Dolan and Moré [2002]) to summarize and compare the runs on the same test suite using different solvers and options. For the easy and moderate problems, we use solution time as a metric for the profiles. For the difficult instances, however, the optimal solution is often not achieved (or even known). For these instances, we use a scaled solution value as the solver metric. We define the scaled solution value of solver s on instance i as $\rho_i^s = 1 + (z_i^s - z_i^*)/z_i^*$, where z_i^s is the best solution value obtained by solver s on instance i, and z_i^* is the best known solution value for instance i. The performance profile therefore indicates the quality of the solution found by a solver within four hours of CPU time.

We start by benchmarking a straightforward implementation of LP/NLP-BB (Algorithm 4) against MINLP-BB, a branch-and-bound algorithm that uses (NLPR(l, u)) to obtain a lower bound. The straightforward LP/NLP-BB implementation does not use any of MINTO's advanced MILP features, such as primal heuristics, cuts, and preprocessing. The algorithm branches on the the most fractional variable and selects the next node to be solved as the one with the smallest lower bound. The implementation is thus quite similar to one created in the Ph.D. work of Leyffer [1993]. We refer to this version of FilMINT as the *vanilla* version.

The performance profiles in Figures 2.2–2.4 compare the performance of the vanilla and MINLP-BB solvers for the easy, moderate and difficult instances in the test suite. For easy instances, the performance of FilMINT and MINLP-BB is quite similar. We drop these instances from further analysis. However, detailed computational results for easy instances are provided in Table A.4 in the Appendix. The results for

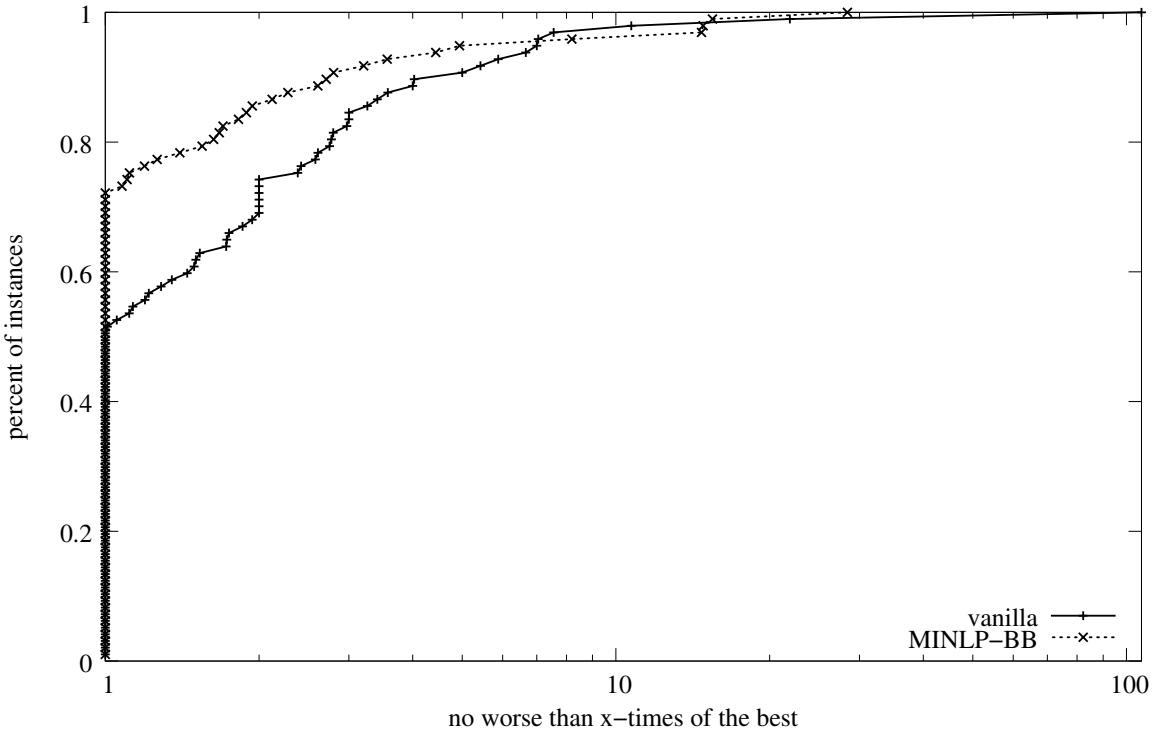

Figure 2.2: Performance Profile Comparing vanilla and MINLP-BB for Easy Instances.

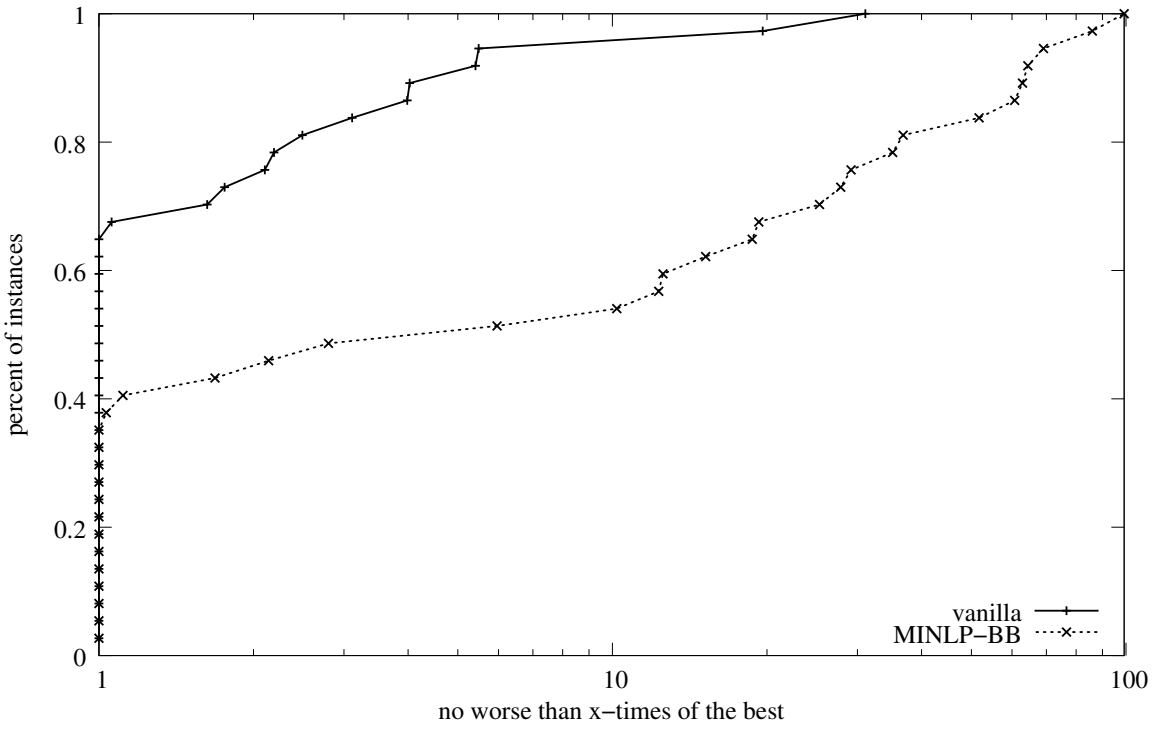

Figure 2.3: Performance Profile Comparing vanilla and MINLP-BB for Moderate Instances.

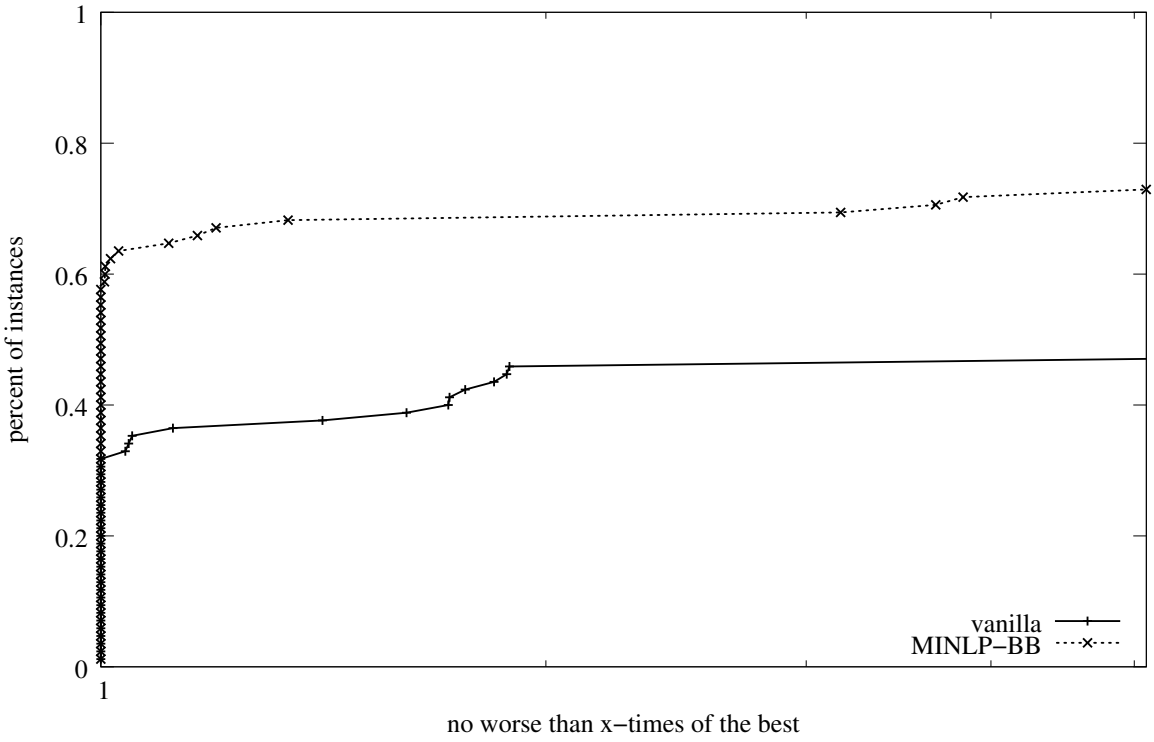

Figure 2.4: Performance Profile Comparing vanilla and MINLP-BB for Difficult Instances.

the moderate problems show a significant improvement of LP/NLP-BB compared to MINLP-BB. The results for the difficult instances, however, show that this simplistic implementation of LP/NLP-BB is not competitive with MINLP-BB for difficult instances, especially for finding high-quality feasible solutions. This observation motivates us to explore the use of advanced MILP features that can easily be switched on with MINTO.

The remainder of our computational experiment is divided into two parts. In the first part, we explore the effect of various MILP features such as cutting planes, heuristics, branching and node selection rules, and preprocessing. By turning on each feature individually, we obtain an indication of which MILP techniques have the biggest impact. The IP features that are found to work well in this part are then included in an intermediate version of our solver (called vanIP). In the second

part, we build on this improved version of LP/NLP-BB by adding features that affect the generation and management of cuts and outer approximations. Each additional feature that appears to improve the performance is included in our final solver, called FilMINT. Finally, we benchmark FilMINT against two MINLP solvers, MINLP-BB (Leyffer [1998]) and Bonmin (Bonami et al. [2008]).

2.2 Exploiting the MILP Framework

In this section we explore the benefits for the LP/NLP-BB algorithm of including standard MIP features such as cutting planes, heuristics, branching and node selection rules, and preprocessing. We conduct careful experiments to assess the impact of these features on the performance of the algorithm.

2.2.1 Cutting Planes, Preprocessing, and Primal Heuristics

Cutting planes have become an important tool in solving mixed-integer programs. FilMINT uses the cut generation routines of MINTO to strengthen the formulation and cut off the fractional solution. If a fractional solution to the linear program $\mathrm{CMP}(\mathcal{K}, l, u)$ is obtained, MINTO tries to exclude this solution with clique inequalities, implication inequalities, lifted knapsack covers, lifted GUB covers, and lifted simple generalized flow covers. The interested reader may examine the survey paper [Linderoth and Ralphs, 2005] and the references therein for a description of these cutting planes. We note that it is unlikely that the linearizations that are added during the course of the tree search will have the required special structure needed for these inequalities to be effective. However, these inequalities can be useful when the linear constraints in the problem, if any, have the desired structure.

Another effective technique in modern algorithms for solving MILPs is preprocessing. By preprocessing the MILP, matrix coefficients and variable bounds can be improved, thereby increasing the bound obtained from the solution to the problem's relaxation. Savelsbergh [1994] explains the variety of preprocessing techniques used for solving MILPs. Many of these same techniques can be applied to the outer-approximation master problem MP($\mathcal{K}$). However, "dual" preprocessing techniques, which rely on inferring characteristics about an optimal solution, may not be valid when applied to MP($\mathcal{K}$) unless all integer points in $Y \cap \mathbb{Z}^p$ are included in $\mathcal{K}$. We therefore deactivate the dual processing features of MINTO in the implementation of FilMINT.

Primal heuristics for MILP aim to find good feasible solutions quickly. A high-quality solution, and the resultant upper bound on the optimal solution value, that is obtained at the beginning of the search procedure can significantly reduce the number of nodes that must be evaluated to solve a MILP. Feasible (integer-valued) solutions to the outer-approximation master MP($\mathcal{K}$) are doubly important to the LP/NLP-BB algorithm, since it is at these points where the problem (NLP(y^k)) is solved, resulting in additional linearizations that are added to MP($\mathcal{K}$). MINTO uses a diving-based primal heuristic to obtain feasible solutions at nodes of the branch-and-bound tree. In the diving heuristic, some integer variables are fixed and the linear program re-solved. The fixing and re-solving is iterated until either the an integral solution is found or the linear program becomes infeasible. FilMINT uses MINTO's primal heuristic to find feasible solutions to (MP($\mathcal{K}$)). We will investigate new techniques for finding feasible solutions for (MINLP) in Chapter 3.

To quantify the effectiveness of MILP cutting planes, preprocessing, and primal heuristics in the context of solving MINLP, we conducted experiments in which the

vanilla LP/NLP-BB algorithm was augmented with each of the individual techniques. Figures 2.5–2.6 show the performance profiles for this experiment. In the figures, the lines MILPcuts, preprocess, and primal-heuristic graph the relative performance of the algorithm with only that feature enabled. The performance of the vanilla version of the algorithm is also depicted.

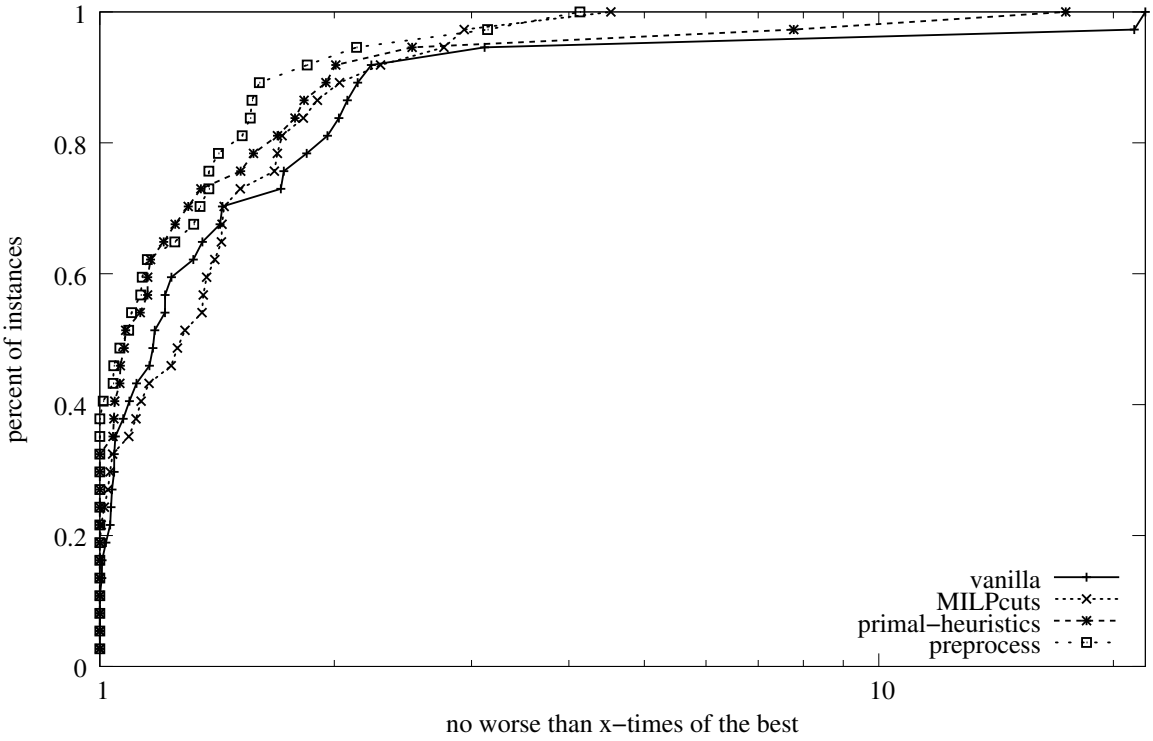

Figure 2.5: Performance Profile Comparing the Effect of MILP cuts, Preprocessing and Heuristics for Moderate Instances.

The results of the experiment indicate that the addition of each individual MILP technique provides a slight improvement over the vanilla algorithm on moderately difficult instances. The results for MILP-based cutting planes confirm the intuition that there is very little special linear structure in the instances under investigation. Table 2.1, that summarizes the results comparing the solvers vanilla and MILPcuts for some instances, shows that MILP cutting planes can be effective when there is sufficient linear structure in the problem. In the table, column 1 shows the instance

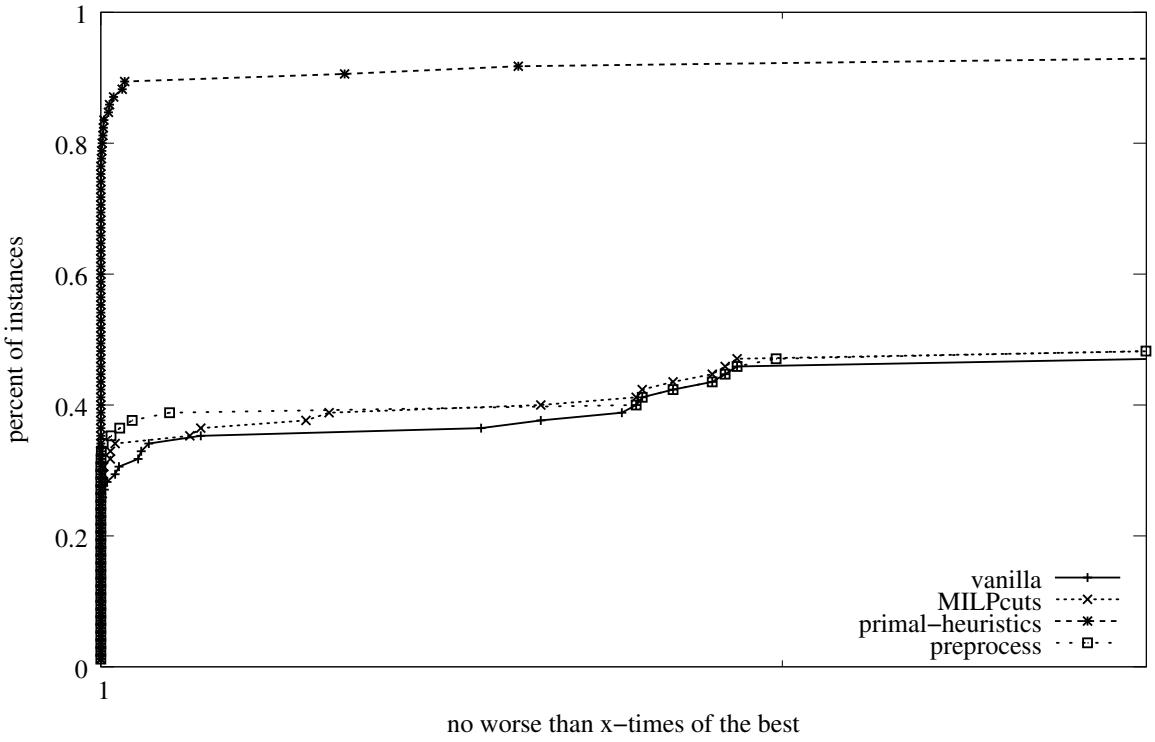

Figure 2.6: Performance Profile Comparing the Effect of MILP cuts, Preprocessing and Heuristics for Difficult Instances.

name, column 2 shows the number of linear constraints in the original formulation, columns 3 and 4 show the time taken and the number of nodes enumerated by vanilla, column 5 shows the number of MILP cuts generated, and columns 6 and 7 show the time taken and the number of nodes enumerated by the solver MILPcuts. For these instances, knapsack covers and flow cover inequalities have a significant effect on the solution scheme. Figure 2.6 shows that the addition of primal heuristics has a very positive impact when solving the difficult instances in the test suite. Recall that our solution metric for the difficult instances was the (scaled) value of the best solution found on that instance, so it is not too surprising that primal heuristics performed so well in this case. It is, however, encouraging to note that heuristics designed to find integer feasible solutions to MILP can be applied on the linear master problem (MP($\mathcal{K}$)), and are often useful for finding high quality solutions to the MINLP. An

emerging line of research is involved in developing primal heuristics especially tailored to MINLP [Bonami et al., 2006], and we extend this line of research in Chapter 3.

Table 2.1: Summary of results for MILP cutting planes for some instances

		vanilla		MILPcuts		
Instance	#L. Cons	Time	Nodes	#MILP Cuts	Time	Nodes
BNS121612	416	13.19	1340	211	8.86	605
Syn10M03M	324	21.28	2370	10	10.86	1398
Syn10M04M	492	59.47	4267	15	49.05	3328
SLay06M	135	39.38	6504	71	18.38	3353

2.2.2 Branching and Node Selection Rules

Another advantage of building FilMINT within the MINTO framework is that it provides the same branching and node selection rules that MINTO provides. A branching scheme is specified by two rules: a branching variable selection rule and a node selection rule, and MINTO comes equipped with a variety of built-in strategies for each. The branching rules available in MINTO are maximum fractionality, penalty-based, strong branching, pseudocost-based, an adaptive method combining the penalty and pseudocost-based methods, and SOS branching. The node selection rules available in MINTO are best bound, depth first, best projection, best estimate, and an adaptive method that searches depth-first until it is estimated that the current node will not lead to an optimal solution, whereupon the node with the best estimated solution is searched. For the adaptive node selection rule, given some estimate E_0 of the optimal solution, the tree is searched depth-first as long as η (objective of (CMP$(\mathcal{K}, l, u)$)) is lesser than E_0. E_0 is estimated as $E_0 = \min_{i \in \mathcal{L}} E_i$, where E_i is the estimate of the best solution obtainable from a node, and is calculated using pseudo-cost information. The interested reader is referred to the paper of Linderoth and Savelsbergh [1999] for

a description of these rules.

The penalty-based and adaptive branching rules require access to the simplex tableau of the active linear program, and these methods were not available in MINTO for our specific choice of LP solver. The SOS-branching strategy is very problem specific, so we excluded it from consideration as a general branching rule. Thus, the branching rules that are of interest include maximum fractionality, strong-branching, and pseudo-cost based. The node selection rules that we tested were best bound, depth first, best estimate, and the adaptive rule. The vanilla algorithm uses by default maximum fractional branching and the best-bound node selection strategy.

To compare the methods, the vanilla version of the LP-NLP/BB algorithm was run using each branching variable and node selection method on all instances in the test suite. In this experiment, when testing the branching variable selection, the best-bound node selection rule was used, and when testing node selection, the most-fractional variable selection method was used.

Performance profiles of the computational experiments for the different branching rules tested are shown in Figures 2.7 and 2.8. The results show that pseudocost branching outperforms all other rules when used in the LP/NLP-BB algorithm. The performance gains are quite significant, but not unexpected, since similar performance gains over maximum fractional branching are also seen for MILP [Linderoth and Savelsbergh, 1999]. The poor performance of strong branching was surprising. The default implementation of strong branching in MINTO can be quite time-consuming, and the time spent seems to be not worth the effort for our test instances.

Performance profiles of the results for the experiments dealing with node selection are given in Figures 2.9 and 2.10. The experimental results indicate that the adaptive node selection rule gives the biggest improvement over pure best-bound search,

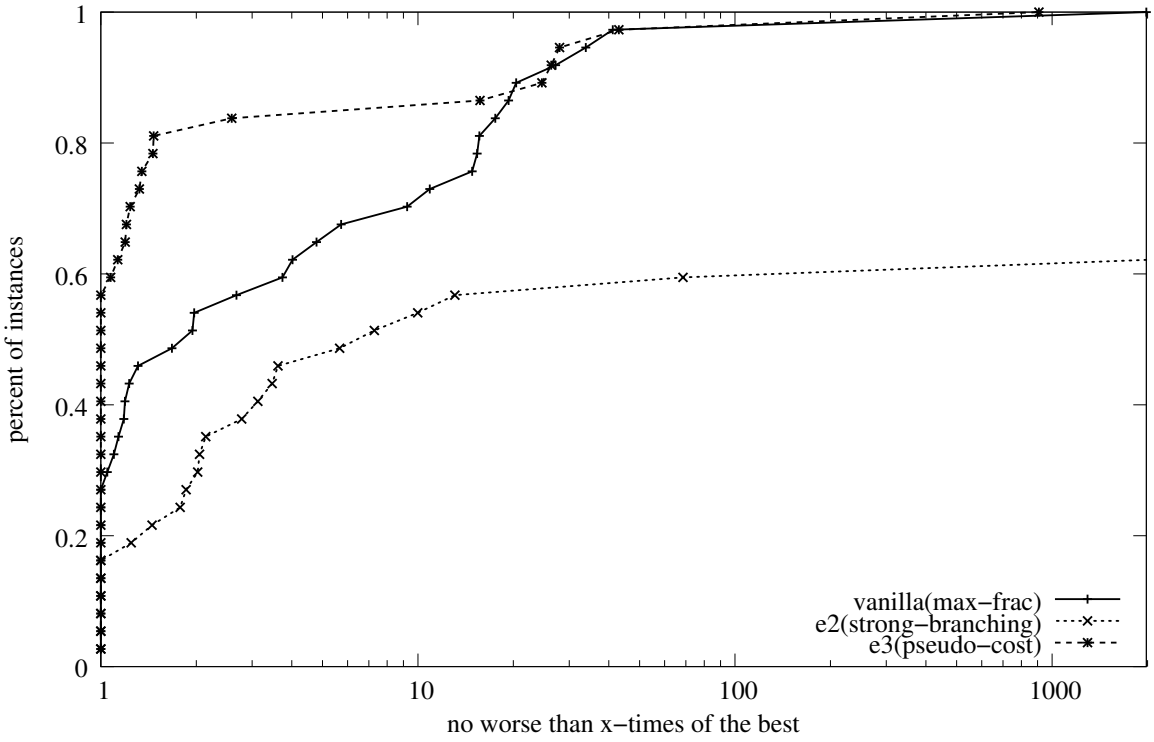

Figure 2.7: Relative Performance of Branching Rules for Moderate Instances.

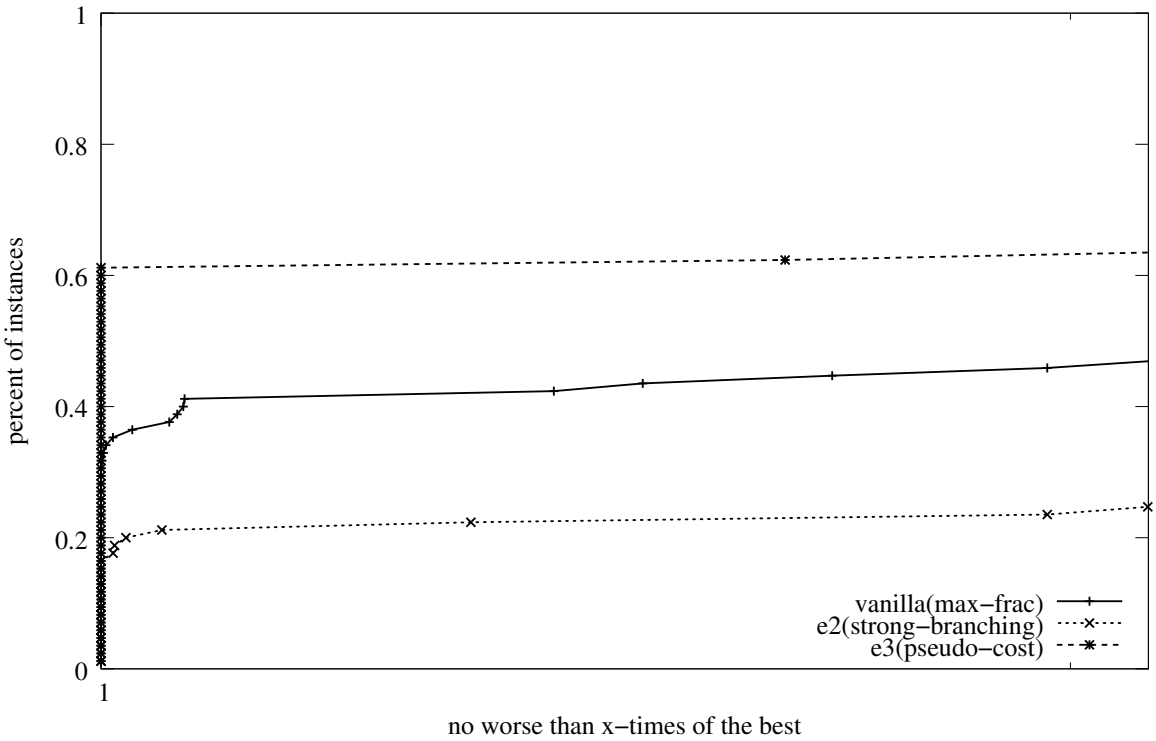

Figure 2.8: Relative Performance of Branching Rules for Difficult Instances.

followed closely by the node selection rule based on best estimates. While the performance gains in terms for the moderate problem instances is good, the improvement for the hard problems is quite significant. This can be explained by the fact that feasible solutions are more likely to be found deep in the branch-and-bound tree, and our metric for hard instances is the scaled value of the best solution found.

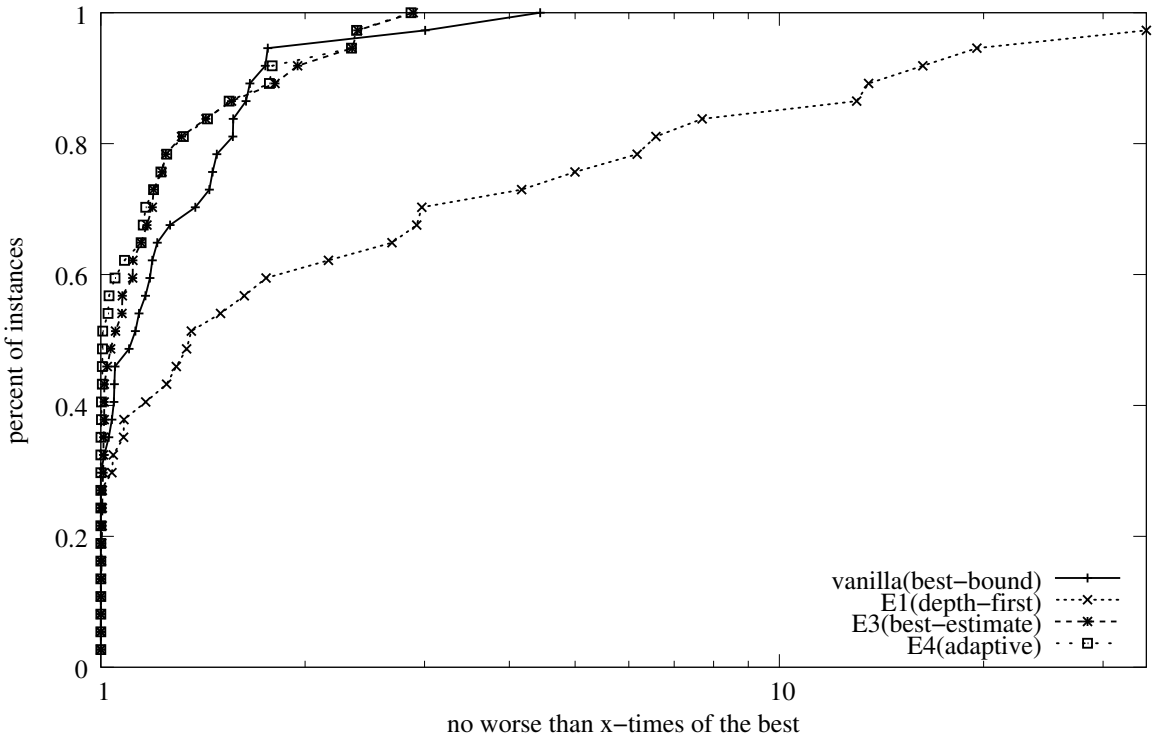

Figure 2.9: Relative Performance of Node Selection Rules for Moderate Instances.

Figures 2.11 and 2.12 compare the relative performance of the MILP-based features for moderate and difficult instances. For moderate instances, pseudocost-based branching has the biggest impact amongst the methods being investigated. For difficult instances, the adaptive node selection method and primal heuristics, followed by pseudocost branching have expectedly, a significant impact in obtaining better integer feasible solutions for the instances. The better performance of adaptive node selection rule compared to primal heuristics for difficult instances can be intuitively explained

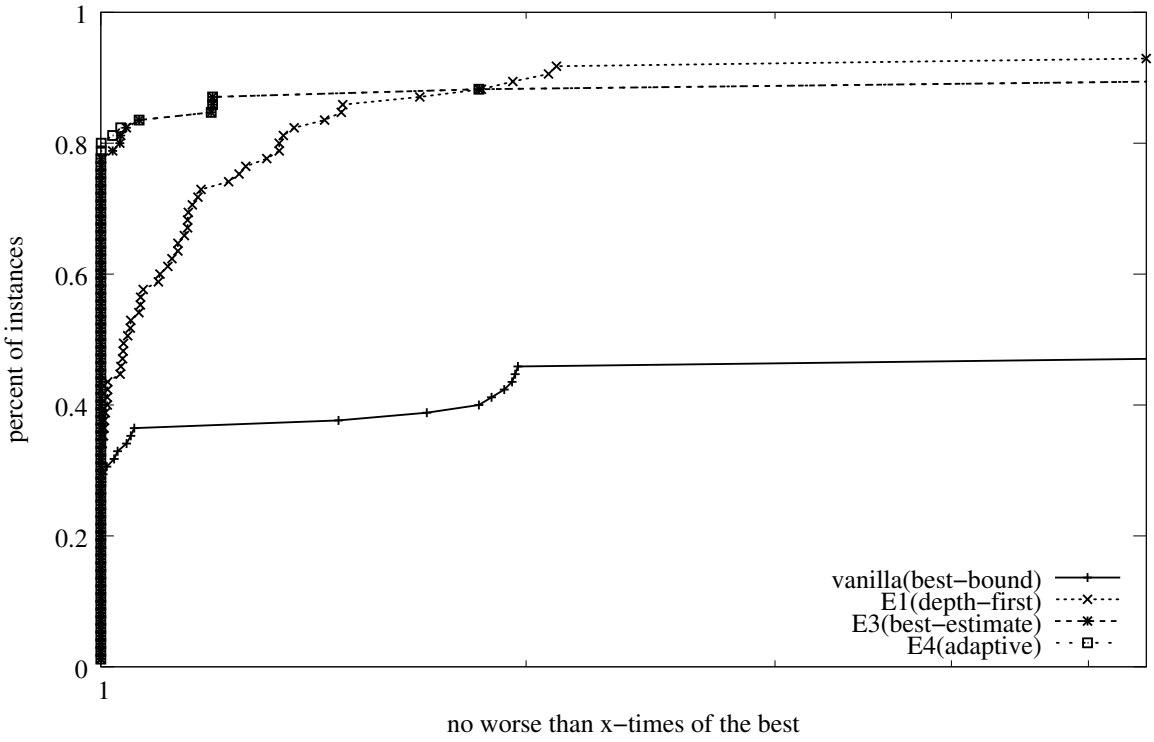

Figure 2.10: Relative Performance of Node Selection Rules for Difficult Instances.

by the fact that the the diving-based primal heuristic obtains an integer solution that is somtimes not feasible for (MINLP). Compared to this, the adaptive node selection method that combines depth-first search with best-estimate search usually provides an integer feasible solution much more quickly during the search. Table 2.2 shows the results for some selected difficult instances comparing the performance of adaptive node selection with primal heuristics. Column 1 in the table refers to the instance name, while columns 2 and 3 show the number of NLPs solved and the number of feasible solutions obtained. Column 4 shows the time taken to obtain the 1^st feasible solution and the NLP which corresponds to the feasible solution. The results show that the adaptive node selection results in a higher percentage of feasible solutions, and this usually results in obtaining better feasible solutions compared to primal heuristics.

Table 2.2: Comparison of Adaptive Node Selection with Primal Heuristics for some difficult instances

	Adaptive node selection				Primal heuristics			
Inst	#NLP	#Feas	1st Sol (time, nlp)	z^*	#NLP	#Feas	1st sol (time, nlp)	z^*
fo7	9	7	201.57 (3)	20.73	11	5	1114.3 (7)	25.64
fo8	11	8	381.85 (3)	29.48	5	1	11270.9 (5)	45.17
m7	6	4	460.83 (2)	106.76	9	6	0.02 (1)	130.46
o7	17	11	191.01 (3)	138.36	7	2	3593.1 (6)	155.93
SL9H	93	93	82.64 (1)	$1.12e^5$	98	94	0.82 (1)	$1.17e^5$
S44M	5	5	0.22 (1)	-876.03	9	9	0.22 (1)	-828.04

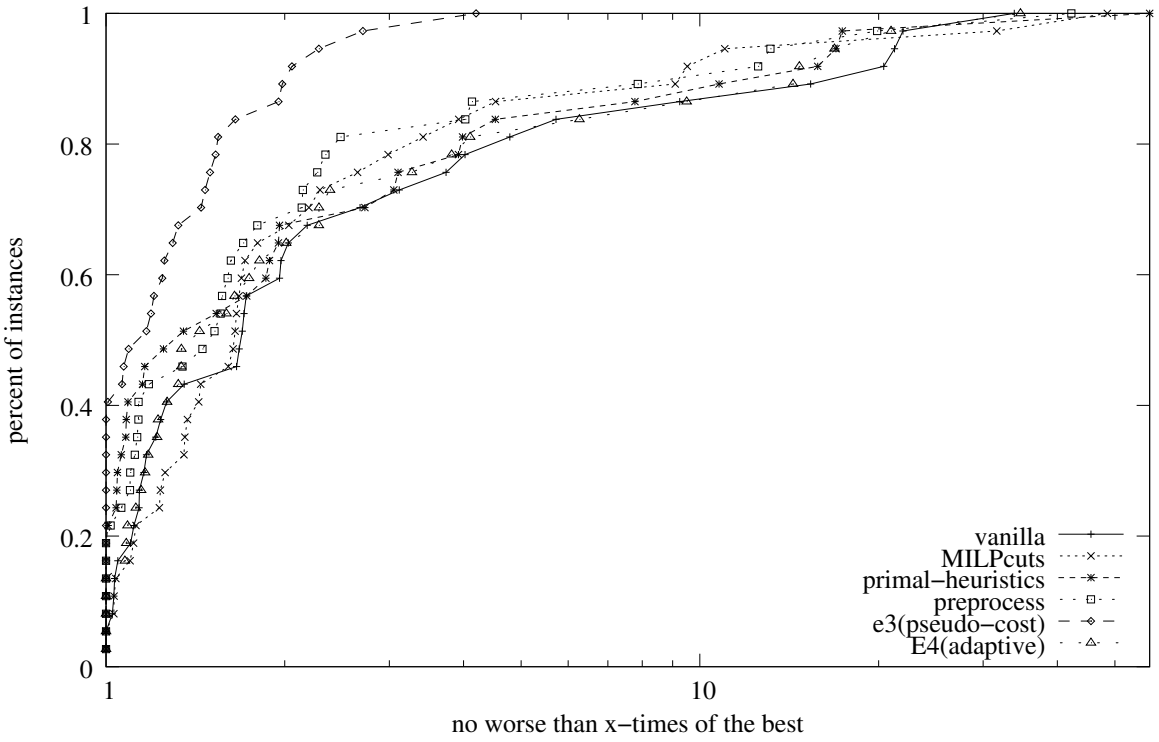

Figure 2.11: Relative Performance of MILP-based features for Moderate Instances.

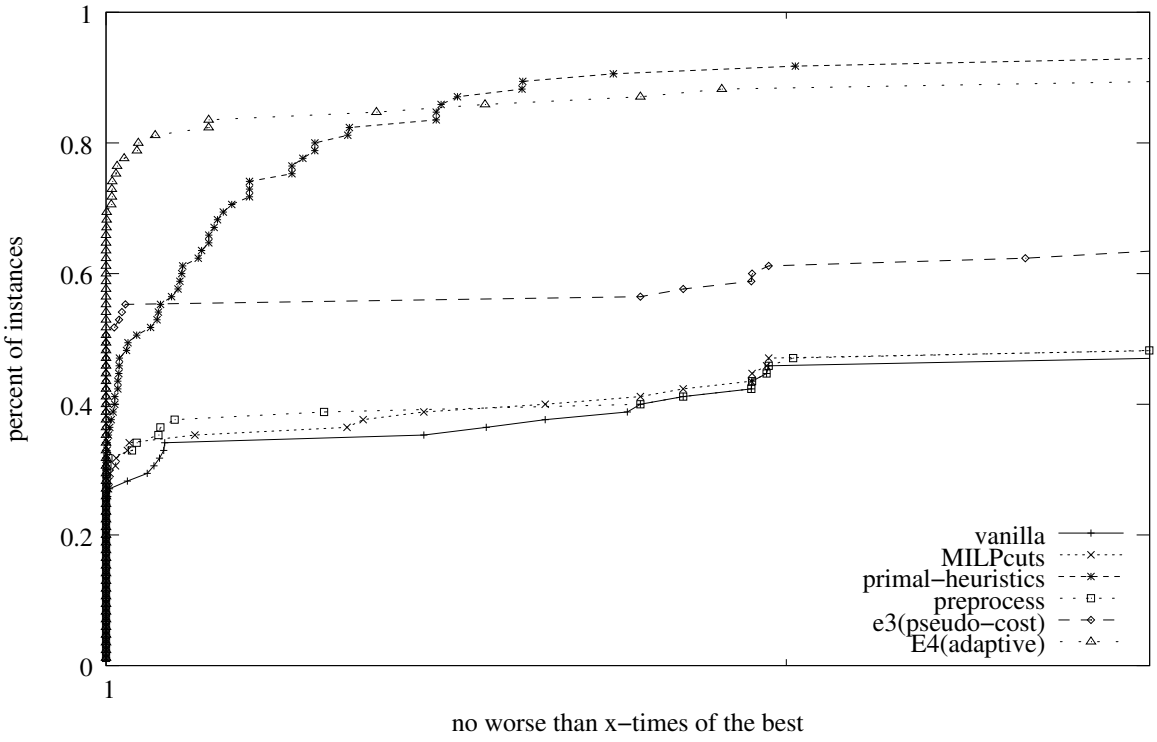

Figure 2.12: Relative Performance of MILP-based features for Difficult Instances.

2.2.3 Summary of MILP Features

The computational experiments helped us identify features from a MILP solver that would improve the efficiency of the LP/NLP-BB algorithm for solving (MINLP). Based on the experiments, we include MINTO's cutting planes, preprocessing, primal heuristics, pseudocost-based branching, and MINTO's adaptive node selection strategy as part of the default solver for subsequent experiments. However, since FilMINT is implemented directly in a MILP branch-and-cut framework, it is a simple matter to implement customized branching rules, node selection strategies, or cutting planes for specific problem classes instead of these default rules.

Figures 2.13–2.14 show the cumulative effect of turning on all selected MILP-based features for both the moderate and difficult instances. In the performance profiles, the label vanIP refers to the solver with the MILP features turned on. The

solver vanIP quite significantly outperforms the straightforward implementation of the LP/NLP-BB solver (vanilla).

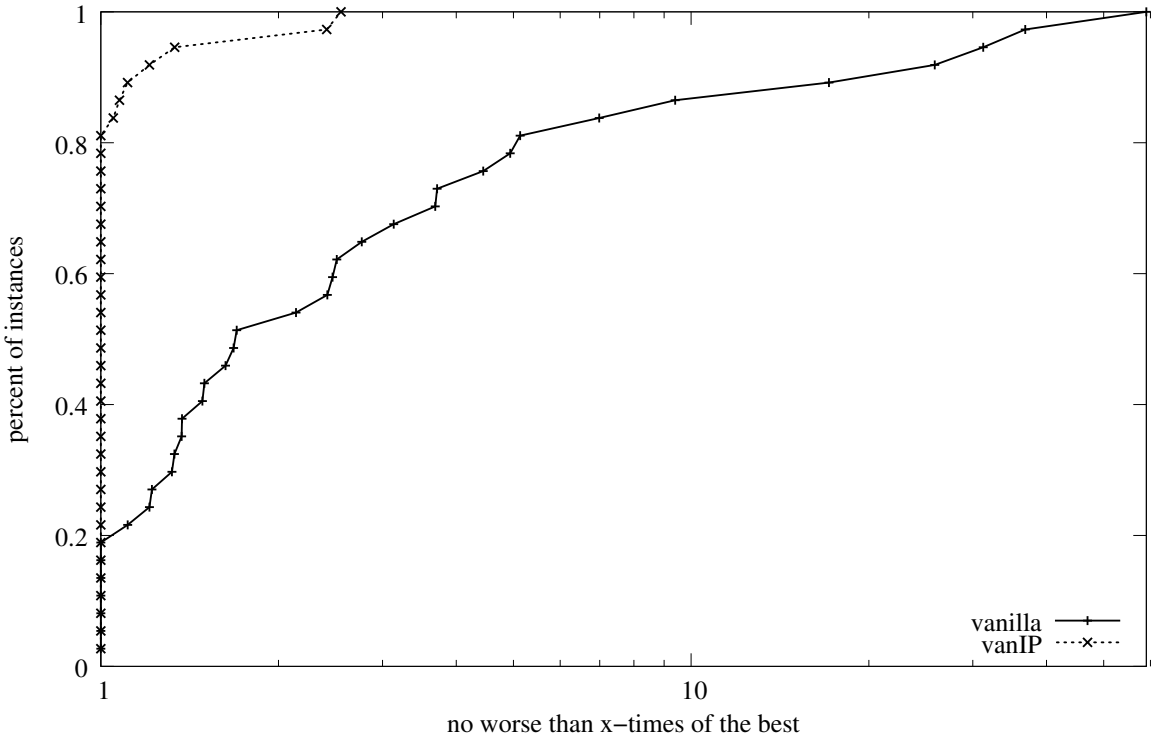

Figure 2.13: Relative Performance of MILP-Enabled Solver for Moderate Instances.

2.3 Linearization Management

In the branch-and-cut algorithm for solving MILP, cuts are used to approximate the convex hull of integer solutions. In the LP/NLP-BB algorithm, linearizations of the constraints are used to approximate the feasible region of the NLP relaxation. For convex MINLPs, linearizations may be generated at *any* point and still give a valid outer-approximation of the feasible region, so we have at our disposal a mechanism for enhancing the LP/NLP-BB algorithm by adding many linear inequalities to the master problem (MP($\mathcal{K}$)). In the branch-and-cut algorithm for MILP, cutting planes like Gomory cuts, mixed-integer-rounding cuts, and disjunctive cuts are similar in

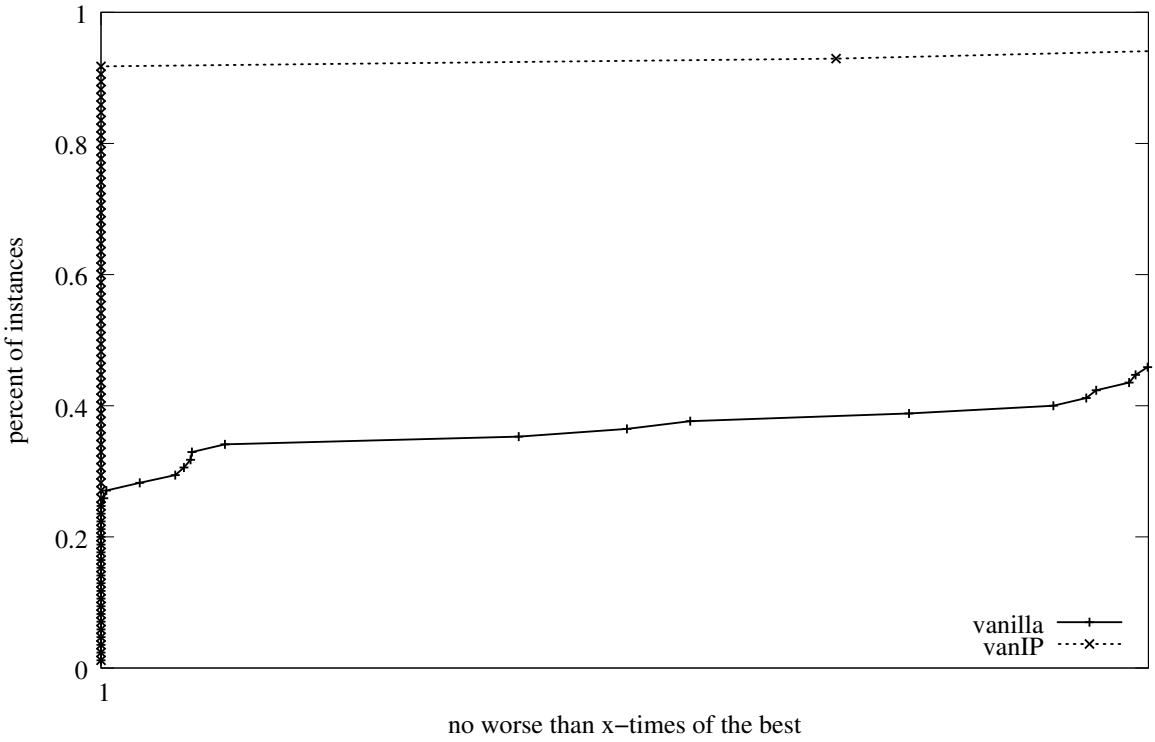

Figure 2.14: Relative Performance of MILP-Enabled Solver for Difficult Instances.

the sense that it is easy to quickly generate a large number of linear inequalities to approximate the convex hull of integer solutions. In our implementation FilMINT, a fundamental design philosophy is to treat linearizations of the nonlinear feasible region in a manner similar to the way cutting planes are treated by a branch-and-cut algorithm for MILP.

In this section, we first discuss simple strategies for effectively managing the large number of inequalities an enhanced LP/NLP-BB algorithm may generate. Key to an effective management strategy is a policy for deciding when inequalities should be added and removed from the master problem (MP($\mathcal{K}$)) and also when additional inequalities should be generated. The section concludes with a discussion of a general mechanism for generating linearizations that trades off the the quality of approximation of the nonlinear feasible region with the time required to obtain the linearization.

2.3.1 Linearization Addition and Removal

Adding linearizations to the master problem (MP($\mathcal{K}$)) increases the solution time of the linear program solved at each node and adds to the storage requirements of the algorithm. An effective implementation must then consider ways to be frugal about adding additional linearizations. Similar implementation issues arise in a branch-and-cut algorithm for MILP, and our techniques are based on this analogy.

One simple strategy for limiting the number of linear inequalities in the continuous relaxation of the master problem (CMP($\mathcal{K}, l, u$)) is to only add inequalities that are violated by the current solution to the linear program. Another simple strategy for controlling the size of (MP($\mathcal{K}$)) is to remove inactive constraints from the formulation. MINTO has a row-management feature that automatically performs this reduction. MINTO monitors the values of the dual variables at the end of every solution to a linear program. If the dual variable for a constraint is zero, implying that the constraint is inactive, for a fixed number of consecutive linear program solutions, then MINTO deactivates the constraint and places the inequality in an auxiliary data structure known as a cut pool. If a constraint in the cut pool later becomes violated, it is added back to the active formulation. MINTO has an environment variable, `MIOCUTDELBND`, which indicates the number of consecutive solutions for which a constraint can be inactive before it is removed. After conducting a few small-scale experiments, the value of `MIOCUTDELBND` was set to 15 in our implementation. Figures 2.15 and 2.16 are performance profiles demonstrating the positive impact of including each of these simple linearization management features in a LP/NLP-BB algorithm for the moderate and difficult instances in our test suite. The label violated refers to the solver in which the feature of adding only violated linearizations is turned on over the solver

vanIP. The label row-mgmt refers to the solver where the solver violated is boosted with MINTO's row management features. While these features helped the solution scheme for moderate instances, they did not have a significant impact for the difficult instances.

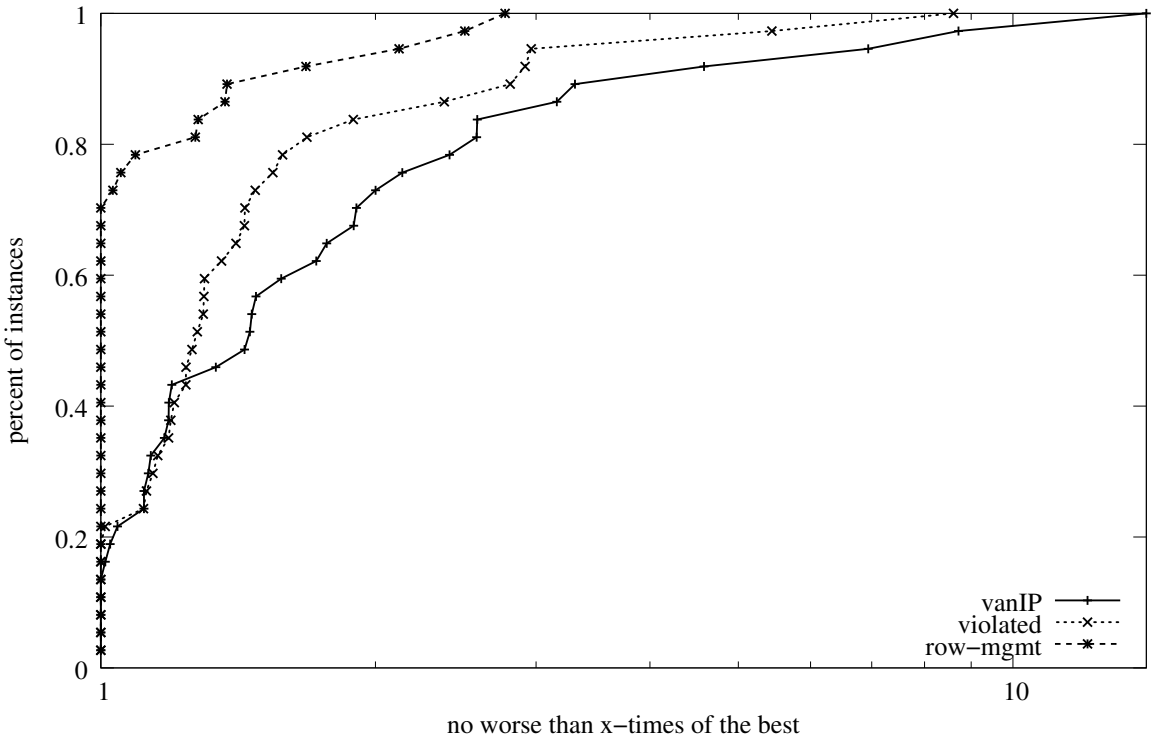

Figure 2.15: Performance Profile Showing the Effect of Row Management for Moderate Instances.

A more sophisticated approach to managing the size of the master problem (MP($\mathcal{K}$)) is to aggregate linearizations obtained from earlier points, rather than removing them. An effective way to perform the aggregation may be based on Benders cuts ([Geoffrion, 1972]). Summing the objective linearizations and the constraint linearizations weighted with the optimal NLP multipliers μ^k from the solution of (NLP(y^k)) yields the Benders cut:

$$\eta \geq f(x^k, y^k) + (\nabla_y f(x^k, y^k)^T + (\mu^k)^T \nabla_y g(x^k, y^k))^T (y - y^k).$$

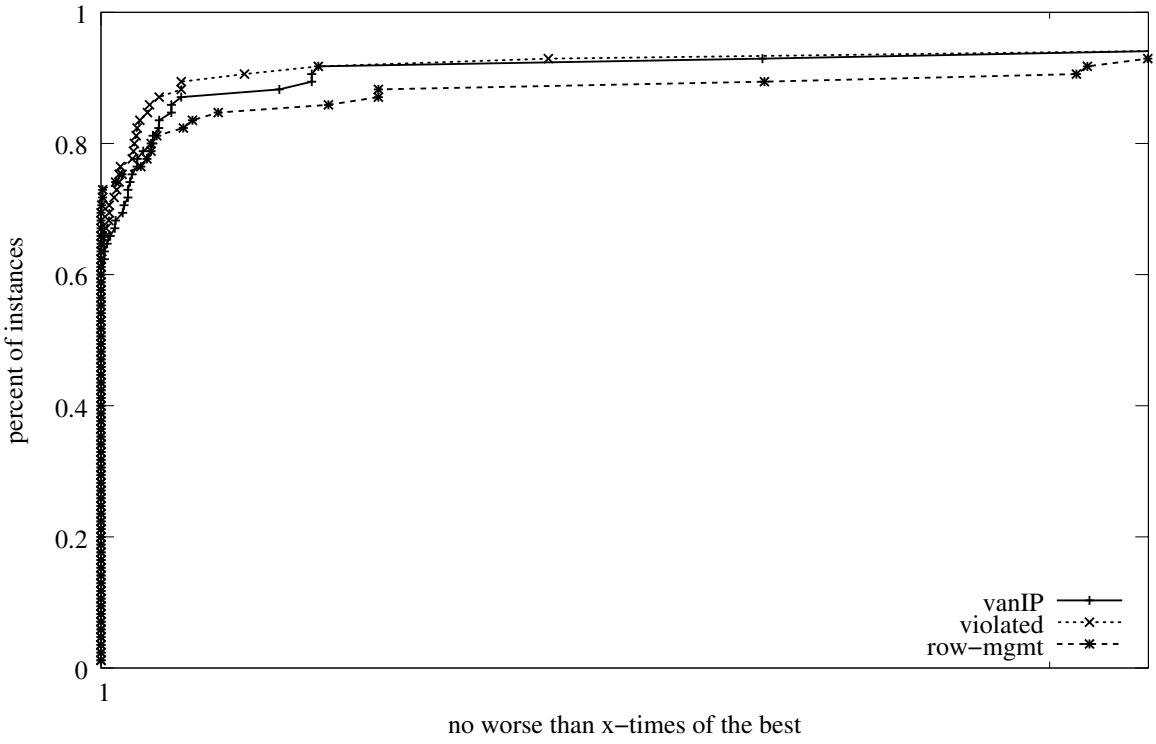

Figure 2.16: Performance Profile Showing the Effect of Row Management for Difficult Instances.

This cut can be simplified by observing that the term

$$\nabla_y f(x^k, y^k)^T + (\mu^k)^T \nabla_y g(x^k, y^k) = \gamma^k$$

corresponds to the NLP multiplier γ^k associated with fixing the integer variables $y = y^k$ in (NLP(y^k)). As explained in Section 1.4.5, the Benders cut is weaker than the outer approximations; on the other hand, it compresses information from $m + 1$ linear inequalities into a single cut. Given MINTO's tunable and automatic row management strategy, we did not see any specific need to employ such aggregation schemes in our first implementation of FilMINT.

2.3.2 Cut or Branch?

In branch-and-cut, there is a fundamental implementation choice that must be made when confronted with an infeasible (fractional) solution: should the solution be eliminated by cutting or branching? Sophisticated implementations of branch-and-cut for solving MILP rely on a variety of *cut management* techniques for answering this question. The interested reader is directed to Atamtürk and Savelsbergh [2005] to see the options for controlling cut management in commercial MILP systems.

One cut management approach typically employed is to not add cutting planes at every node of the search tree. Rather, cutting planes are generated at certain nodes, for example at a fixed interval, with a bias towards generating cuts at nodes close to the root of the search tree. For the LP/NLP-BB algorithm, the branching variable selection provides an additional motivation for generating linearizations very early in the search process. Most branching procedures are based on selecting a variable that will increase the objective function of the two child subproblems significantly. As pointed out by Forrest et al. [1974], branching on a variable that has little or no effect on the solution at subsequent nodes results in a *doubling* of the amount of work necessary to completely process that node. Thus, by the very nature of the branch-and-bound process, the branching decisions made at the top of the tree are the most crucial. In the LP/NLP-BB Algorithm 4, if few linearizations ($\text{OA}(x^k, y^k)$) are included in the master problem ($\text{MP}(\mathcal{K})$), the problem can be a very poor approximation of the true problem (MINLP), and branching decisions made based on this master problem may be very poor. Thus, it is quite important for the LP/NLP-BB algorithm that the master problem ($\text{MP}(\mathcal{K})$) obtain good linearization information early in the solution process.

A second cut management technique is to add cuts for multiple iterations (typically called *rounds*) at a given node. Algorithmic parameters control the number of rounds at a node.

Based on the analogy to cut management in branch-and-cut for MILP, there are three parameters that we use to control the linearization strategy of our implementation of the LP/NLP-BB algorithm. The first parameter β controls the likelihood that linearizations will be generated at a node. The strategy is biased so that linearizations are more likely to be generated at nodes high in the search tree. Specifically, the probability that linearizations are generated at a node is $\min\{\beta 2^{-d}, 1\}$. Note that in a complete search tree, there are 2^d nodes at level d, so the parameter β is the expected number of nodes at level d at which cuts are generated, if the search tree was complete. The second parameter K is used for detecting "tailing off" of the linearization procedure. If the percentage change in the solution value of the relaxed master problem (CMP$(\mathcal{K}, l, u)$) is not at least K, then the algorithm decides to branch rather than to add more linearizations about the current solution point. The final parameter is responsible for ensuring that there is a reasonable balance between branching and cutting by constraint linearizations. Linearizations will be taken around at most M different points at any one node of the search tree. The parameters (β, K, M) are specific to the *type* of point about which we are generating linearizations, the details of which are explained next.

2.3.3 Linearization Generation Methods

A simple observation is that due to the convexity of the functions f and g in (MINLP), the outer-approximation inequalities (OA(x^k, y^k)) are valid regardless of the point (x^k, y^k) about which they are taken. This gives the LP/NLP-BB algorithm great

flexibility in generating linearizations to approximate the feasible region of (MINLP). The tradeoff to consider is the time required to generate the linearization versus the quality/strength of the resulting linearization. We examined three different ways to select points about which to add linearizations, spanning the spectrum of this tradeoff.

The first method simply linearizes the functions f and g about the fractional point $(\hat{x}, y^k)$ obtained as a solution to (CMP$(\mathcal{K}, l, u)$) at the **evaluate** step of Algorithm 4. This point selection mechanism is called the *ECP-based method*, as it is analogous the Extended Cutting Plane method for solving (MINLP) proposed by Westerlund and Pettersson [1995], which is itself an extension of the cutting plane method of Kelley [1960] for solving convex programs. The ECP-based point selection has the advantage that it is extremely fast to generate linearizations, requiring only the evaluation of the gradient of the objective function and the Jacobian of the constraints at the specified point. However, the points about which linearizations are taken may be far from feasible, and thus the resulting linearizations form a poor approximation to the true nonlinear feasible region.

In the second point selection method, we fix the integer decision variables to the values obtained from the solution of (CMP$(\mathcal{K}, l, u)$) ($y = y^k$), and the nonlinear program (NLP(y^k)) is solved to obtain the point about which to linearize. This method is called *fixfrac*, as it is similar in spirit to the original LP/NLP-based algorithm, but now integer decision variables y may be fixed at *fractional* values. The fixfrac method has the advantage of generating linearization about points that are closer to the feasible region than the ECP-based method, at the expense of solving the nonlinear program (NLP(y^k)).

In the third point selection method, no variables are fixed (save those that are fixed by the nodal subproblem), and the NLP relaxation (NLPR$(l,\ u)$) is solved to

obtain a point about which to generate linearizations. This is called the *NLPR-based method*. The linearizations obtained from these are the tightest of the three methods. However, it can be time-consuming to solve the NLP so as to compute the linearizations.

The ECP, fixfrac, and NLPR methods differ in the set of variables fixed before solving a nonlinear program to determine the linearization point. The ECP-based method fixes *all* variables from the solution of $(\mathrm{CMP}(\mathcal{K}, l, u))$, the fixfrac method fixes only the integer decision variables y, and the NLPR-based method fixes no decision variables.

The three classes of linearizations form a hierarchy, with ECP linearizations being the weakest/cheapest, and NLPR cuts being the strongest/most costly. The manner in which we generate linearizations in FilMINT exploits this hierarchy. First, ECP linearizations are generated until it appears that they are no longer useful. Next, fixfrac linearizations are generated, and finally, NLPR linearizations are generated. Our final strategy for producing points about which to generate linearizations combines all three methods and is given in pseudo-code form in Algorithm 5.

Values for the parameters in Algorithm 5 to use in the default version of FilMINT were chosen after careful experimentation. Figures 2.17–2.18 compare the effect of choosing different values of β for the ECP point selection method, used on top of the solver row-mgmt. The values of β_{ecp} are choosen to be 10, 100, and 1000, and the solvers ecp-10, ecp-100, and ecp-1000 correspond to these choices respectively. Similarly, Figures 2.19–2.20 compare the effect of choosing different values of β for the Fixfrac method, when run on top of the solver row-mgmt. The solver fixfrac-1 corresponds to the choice of $\beta_{\mathrm{ff}} = 1$, and the solver fixfrac-10 corresponds to the choice of $\beta_{\mathrm{ff}} = 10$. In these experiments, the choice of M and K are choosen to be 10 and

Let d = depth of current node (l, u, η).
Let $r \in [0, 1]$ be a uniformly generated random number
if $r \leq \beta_{\text{ecp}} 2^{-d}$ **AND** $(\eta^k - \eta^{k-1})/|\eta^{k-1}| \geq K_{\text{ecp}}$ **AND** $n_{\text{ecp}} \leq M_{\text{ecp}}$ **then**
  $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\hat{x}, y^k)\}$. **(ecp)**
  $n_{\text{ecp}} \leftarrow n_{\text{ecp}} + 1$.
else if $r \leq \beta_{\text{ff}} 2^{-d}$ **AND** $(\eta^k - \eta^{k-1})/|\eta^{k-1}| \geq K_{\text{ff}}$ **AND** $n_{\text{ff}} \leq M_{\text{ff}}$ **then**
  Solve NLP(y^k). **(fixfrac)**
  if NLP(y^k) is feasible **then**
    Let $(\tilde{x}^k, y^k)$ be solution to NLP(y^k)
  else
    Let $(\tilde{x}^k, y^k)$ be solution to NLPF(y^k)
  end if
  $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\tilde{x}^k, y^k)\}$.
  $n_{\text{ff}} \leftarrow n_{\text{ff}} + 1$.
else if $r \leq \beta_{\text{nlpr}} 2^{-d}$ **AND** $(\eta^k - \eta^{k-1})/|\eta^{k-1}| \geq K_{\text{nlpr}}$ **AND** $n_{\text{nlpr}} \leq M_{\text{nlpr}}$ **then**
  Solve NLPR(l^k, u^k). **(nlpr)**
  Let $(\bar{x}^k, \bar{y}^k)$ be solution to NLPR(l^k, u^k)
  $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\bar{x}^k, \bar{y}^k)\}$.
  $n_{\text{nlpr}} \leftarrow n_{\text{nlpr}} + 1$.
end if

Algorithm 5: Algorithm for generating linearizations.

0.1 respectively.

Table 2.3 summarizes the final parameter settings in the default version of FilMINT. Note that $\beta_{\text{nlpr}} = 0$, so that NLPR linearizations are not added at nodes of the branch-and-cut tree by default in FilMINT. However, as stated in Algorithm 4, linearizations about the solution to the original nonlinear programming relaxation NLPR(y^l, y^u) are added to initialize the algorithm.

Table 2.3: Default Linearization Parameters

Method	β	M	K
ECP	10	10	0.1%
Fixfrac	1	10	0.1%
NLPR	0	1	0.1%

To demonstrate the effectiveness of the linearization-point selection methodologies, an experiment was conducted wherein the MILP-technique enhanced version

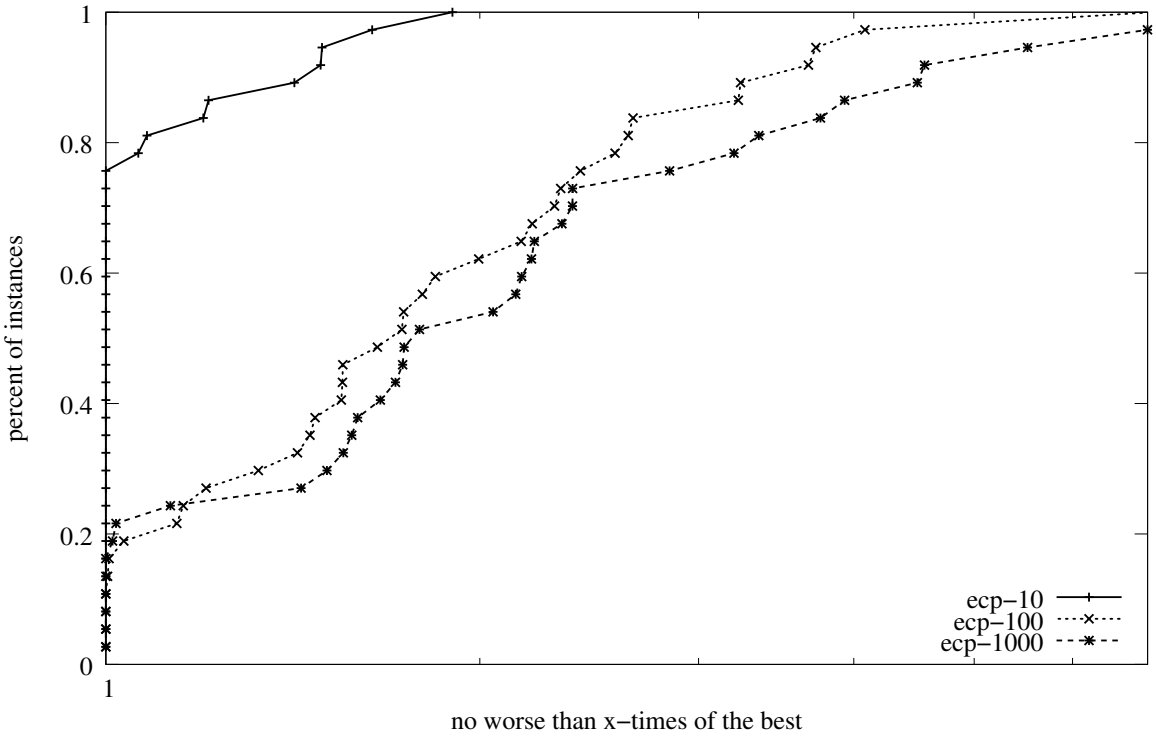

Figure 2.17: Performance Profile Comparing Different Parameter Choices for ECP on Moderate Instances

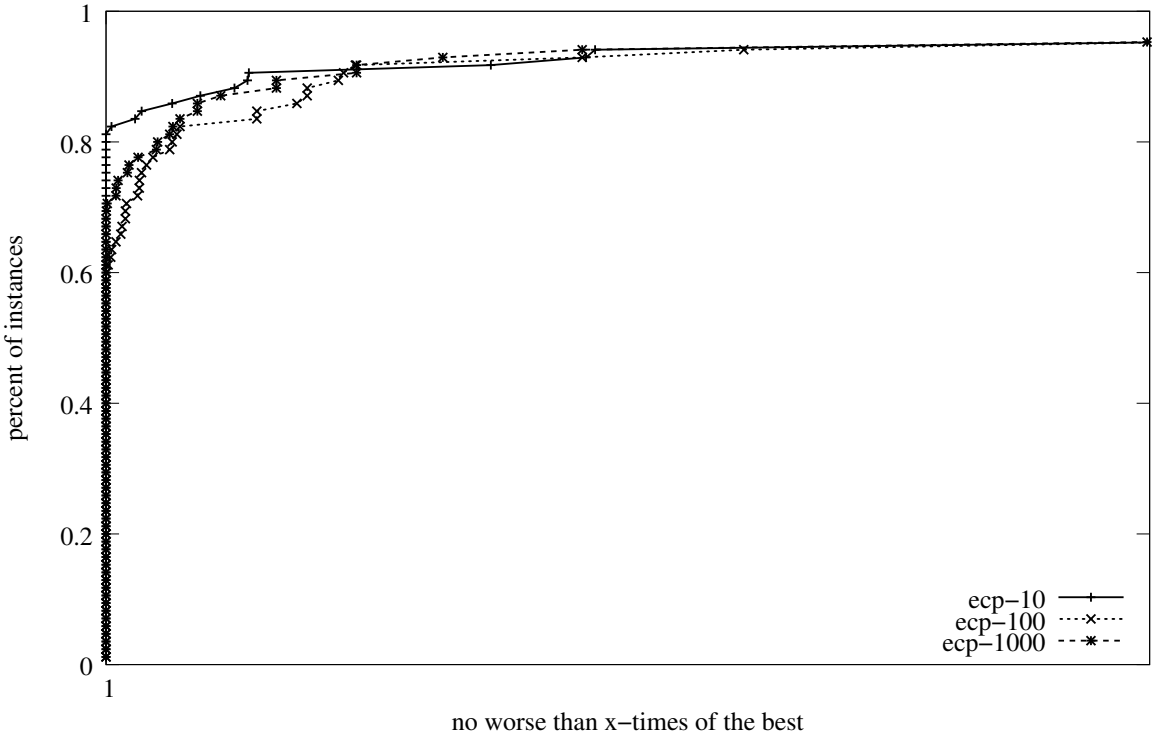

Figure 2.18: Performance Profile Comparing Different Parameter Choices for ECP on Difficult Instances

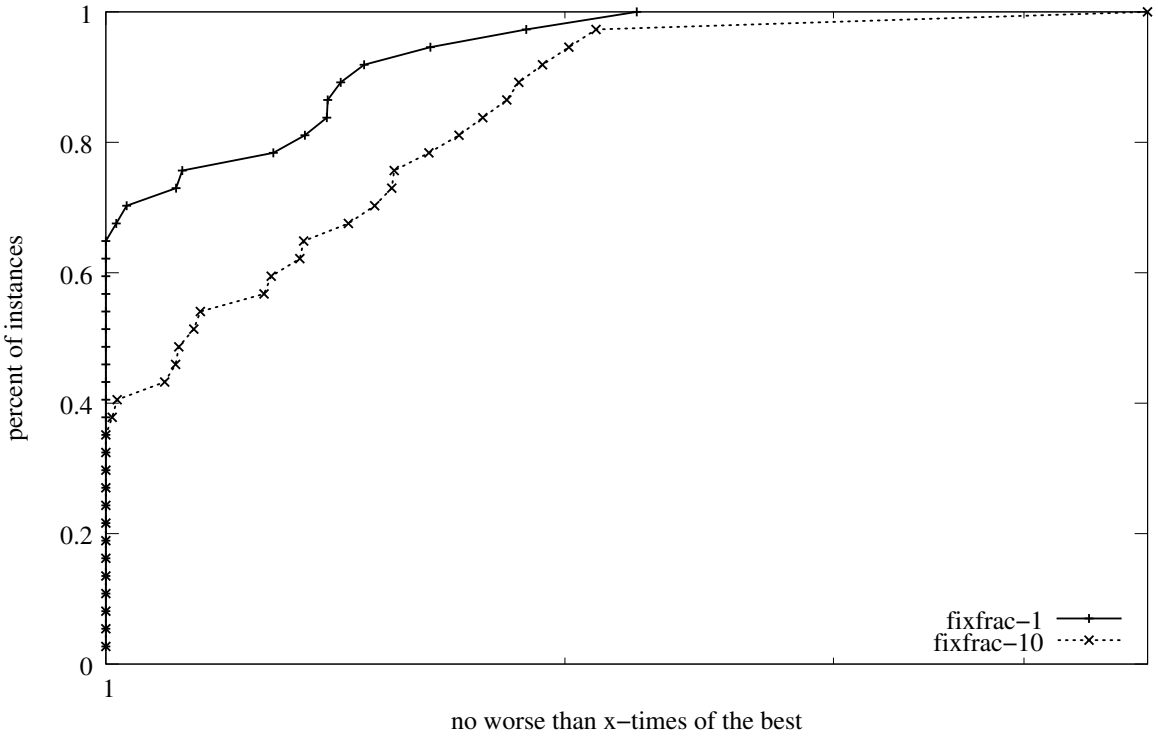

Figure 2.19: Performance Profile Comparing Different Parameter Choices for Fixfrac on Moderate Instances

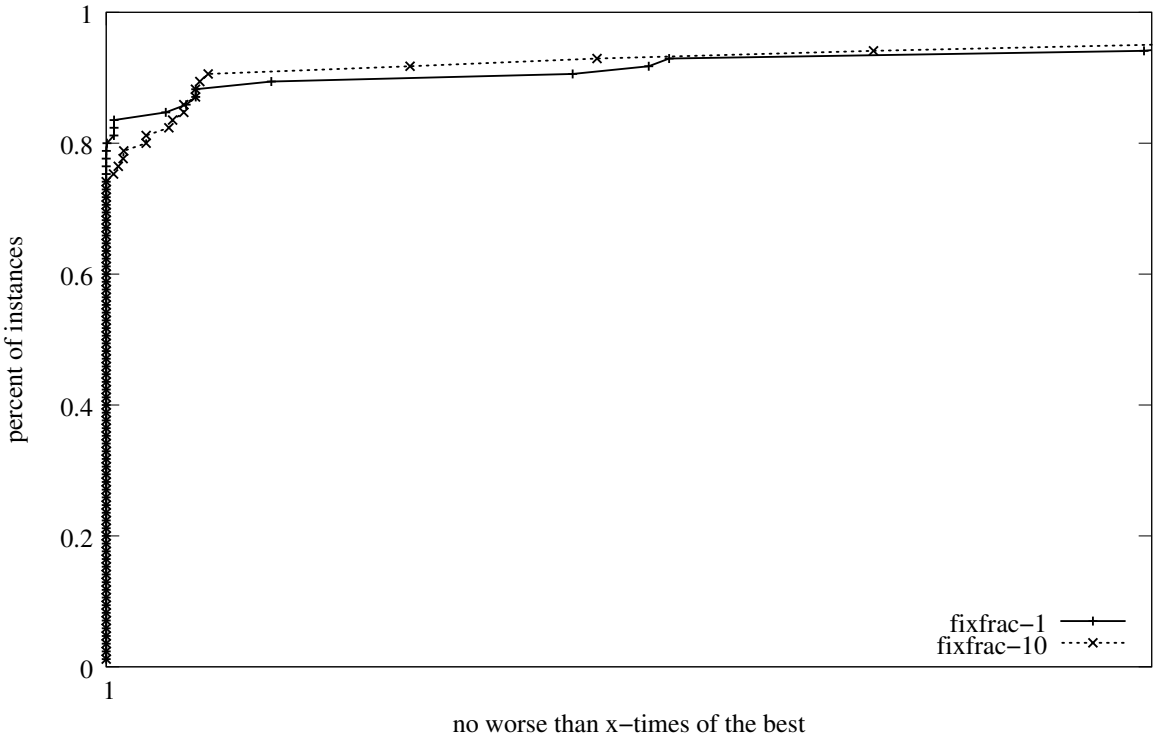

Figure 2.20: Performance Profile Comparing Different Parameter Choices for Fixfrac on Difficult Instances

of the LP/NLP-BB algorithm was further augmented with each of the additional linearizations techniques individually. Runs of the experiment can be viewed as implementing Algorithm 5, with $M = 0$ for the linearization point selectiom mechniams *not* under study. The cut management parameters (β, M, K) for the chosen linearization point selection method were the same as in Table 2.3, except for NLPR, in which a value of $\beta_{\text{nlpr}} = 1$ was used. Figures 2.21 and 2.22 show performance profiles of this experiment. The solver row-mgmt on the profile is the MILP-enhanced LP/NLP-BB algorithm without any additional linearizations, but has MINTO's row management and the option to add only violated inequalities turned on. The solver filmint is the default version of the FilMINT solver. The experiment conclusively demonstrates that linearizing about additional points is quite advantageous for the LP/NLP-BB algorithm. Also, using multiple mechanisms for choosing the points about which to generate linearizations, as is done in FilMINT, helps the algorithm's performance.

2.4 Comparison of MINLP Solvers

In this section, we report on an experiment comparing the performance of FilMINT with two other well-known solvers for MINLPs on our suite of test instances. FilMINT is compared to Bonmin [Bonami et al., 2008] and to MINLP-BB [Leyffer, 1998]. MINLP-BB is a branch-and-bound solver based that uses the nonlinear programming relaxation NLPR(l, u) to provide a lower bound on z_{MINLP}. The NLP subproblems in MINLP-BB are solved by filterSQP, the same solver that is used in FilMINT. The Bonmin solver consists of a suite of algorithms, and FilMINT is compared against the `I-Hyb` hybrid algorithm of Bonmin. The `I-Hyb` algorithm of Bonmin is an implementation of the LP/NLP-BB algorithm that has been augmented with two additional

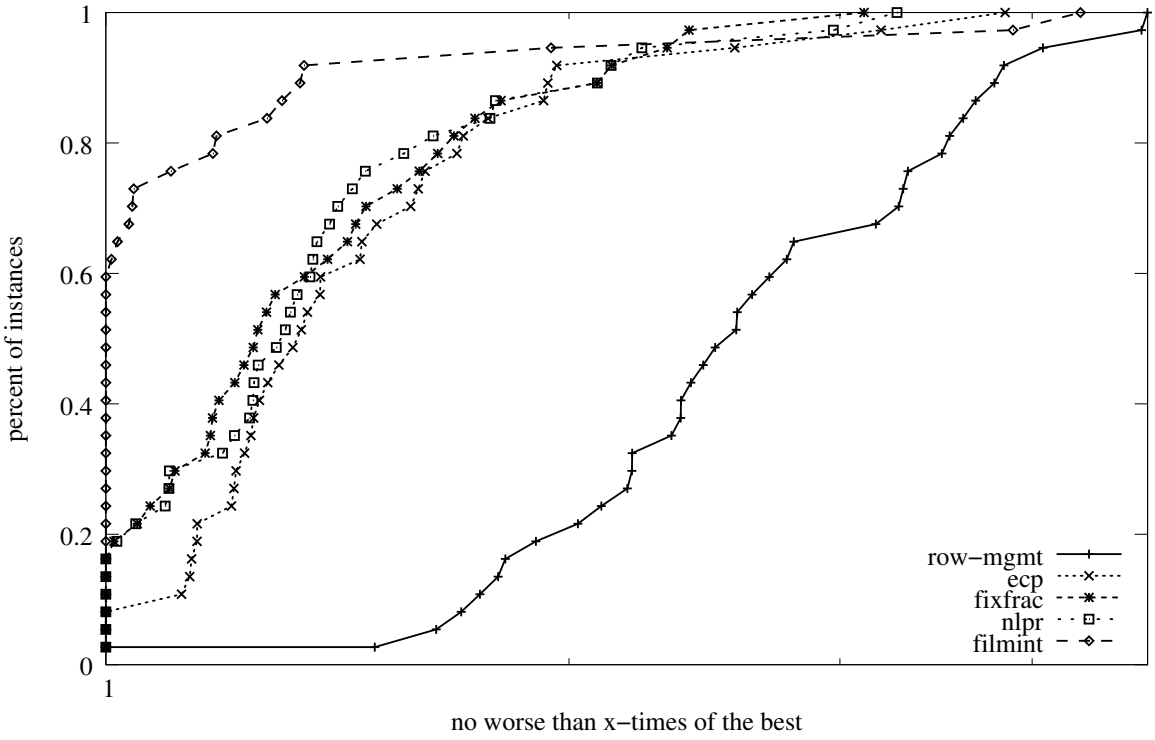

Figure 2.21: Performance Profile Comparing Linearization Point Selection Schemes on Moderate Instances

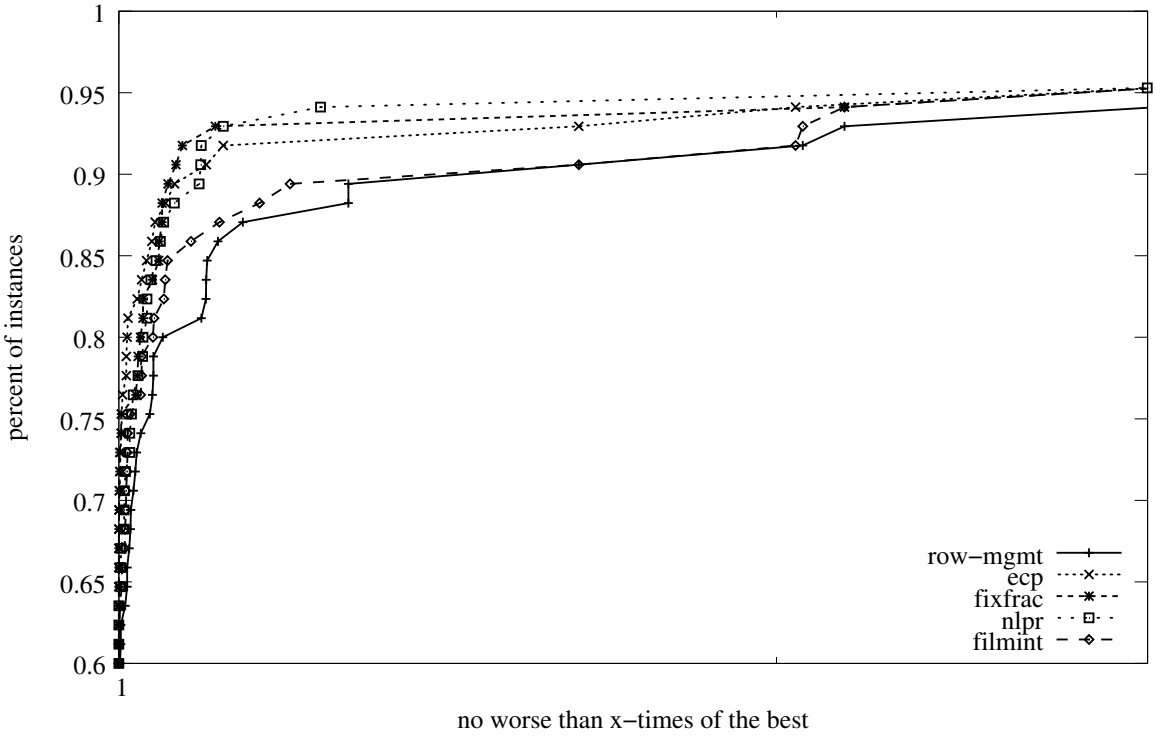

Figure 2.22: Performance Profile Comparing Linearization Point Selection Schemes on Difficult Instances

features. First, NLP relaxations NLPR(l, u) are solved occasionally (every $L = 10$ nodes) during the branch-and-bound search. Second, truncated MILP branch-and-bound enumerative procedures are performed in an effort to find integer-feasible solutions. The enumerative procedure performed by Bonmin is akin to the diving-based primal heuristic found in MINTO v3.1 (and used by FilMINT).

Experiments were run using two different versions of the Bonmin software. The first version `Bonmin` is a version of the executable built from source code taken from the COIN-OR code repository during the summer of 2006. The second version, `Bonmin-v2` refers to a more recent release of the software, (specifically version 0.1.4), available from `https://projects.coin-or.org/Bonmin/browser/releases`.

The parameters in the default version of FilMINT were optimized based on all previous experiments. Specifically, MINTO v3.1's default MIP features (preprocessing, pseudocost branching variable selection, adaptive node selection, cutting planes) were enabled. Linearizations in default FilMINT are generated, added, and removed in a manner described in Section 2.3, with the paramaters given in Table 2.3.

Figures 2.23–2.25 compare the solvers FilMINT, Bonmin (v1), Bonmin (v2), MINLP-BB, and the vanilla implementation of the LP/NLP-BB algorithm. Also, for difficult instances, Figure 2.26 is a time-based performance profile that compares the different solvers. The computational setup used for this experiment is the same that we have used for all our experiments, detailed in Section 2.1.2. All solvers were given a time limit of four hours for each instance. The profile reveals that for easy instances, FilMINT is clearly the most effective solver. For the moderate instances, FilMINT and Bonmin-v2 are clearly the most effective solvers. Detailed results showing the exact times taken to solve the instances for the different solvers are also provided in the Appendix in Table A.5. For the difficult instances, FilMINT and both versions

of Bonmin are the most effective at finding high quality feasible solutions within the four-hour time limit. Tables showing the best solution value obtained at the end of the run for each instance are given in the Appendix (Table A.6). A total of 88 out of the 122 instances in our test suite are solved by one of the 5 solvers to provable optimality within the 4 hour time limit. Figure 2.27 is a performance profile (using time as the performance metric) comparing each solver's performance on this subset of the instances.

Detailed results for all instances in the testsuite run using FilMINT (with all features turned on) as the solver are provided in Tables A.7–A.18. Tables A.7–A.9 show the statistics related to the number of nodes evaluated, number of LPs solved, number of integer solutions obtained, number of feasible solutions obtained, and the number of constraints generated, including the MILP-based cuts. The results show that the number of MILP-based cuts is usually very small compared to the linearizations generated for cutting off fractional solutions. This is primarily because of the fact that the linear part of the formulation for these instances often do not contain the special structure that is needed for the generation of these MILP-based cuts. Further, we now see that a high percentage of integer solutions generated are actually feasible, except in the case of some very difficult instances like the trimloss set of instances.

Tables A.10–A.15 show the statistics related to the solution of various NLPs investigated in this Chapter. More specifically, the tables show the number of NLPs solved, the average time taken to solve an NLP, and the number of linearizations added to (MP($\mathcal{K}$)) during the course of the tree search. The different NLPs that are solved are (NLP(y^k)) for an integer y^k, (NLP(y^k)) for a fractional y^k (Fixfrac method), and (NLPR(l, u)). We also include the statistics for the ECP method in

these tables. The results show that generating linearizations by the ECP methods is the least expensive, compared to the NLPR method or the Fixfrac method. Also, the average time taken to solve (NLP(y^k)) for the fixfrac method is much lesser than the time needed to solve (NLPR(l, u)).

Finally, Tables A.16–A.18 show the time-related statistics for the runs with FilMINT as the solver. The tables show the total time taken for each instance, the time spent in solving LPs and NLPs, the time spent in heuristics for obtaining integer solutions, and also the time spent in generating linearizations and searching cuts from the cut pool of MINTO. The results show that on an average, more than 90% of the time is spent in solving LPs. Also, the time spent in solving NLPs is much less than 5% at best for most instances. The fact that most of the time is spent in solving LPs and doing the tree search, and that the solver FilMINT does quite well compared to other solvers for our testsuite of instances, justifies our design of the framework for solving MINLPs by exploiting the MILP-based features and adding more linearizations. This also addresses our goal of coming up with an efficient framework that solves MINLPs at a cost which is a small multiple of the cost of solving comparable MILPs.

2.5 Conclusions

We introduced an algorithmic framework that aims to solve convex mixed integer nonlinear programs at a cost which is only a small multiple of the cost of solving a comparable MILP. Our new solver, FilMINT, is an implementation of the Quesada-Grossmann LP/NLP-BB algorithm for solving convex MINLPs in a branch-and-cut framework. We augment FilMINT with modern algorithmic techniques found in mixed integer programming software, by identifying their impact in the solution

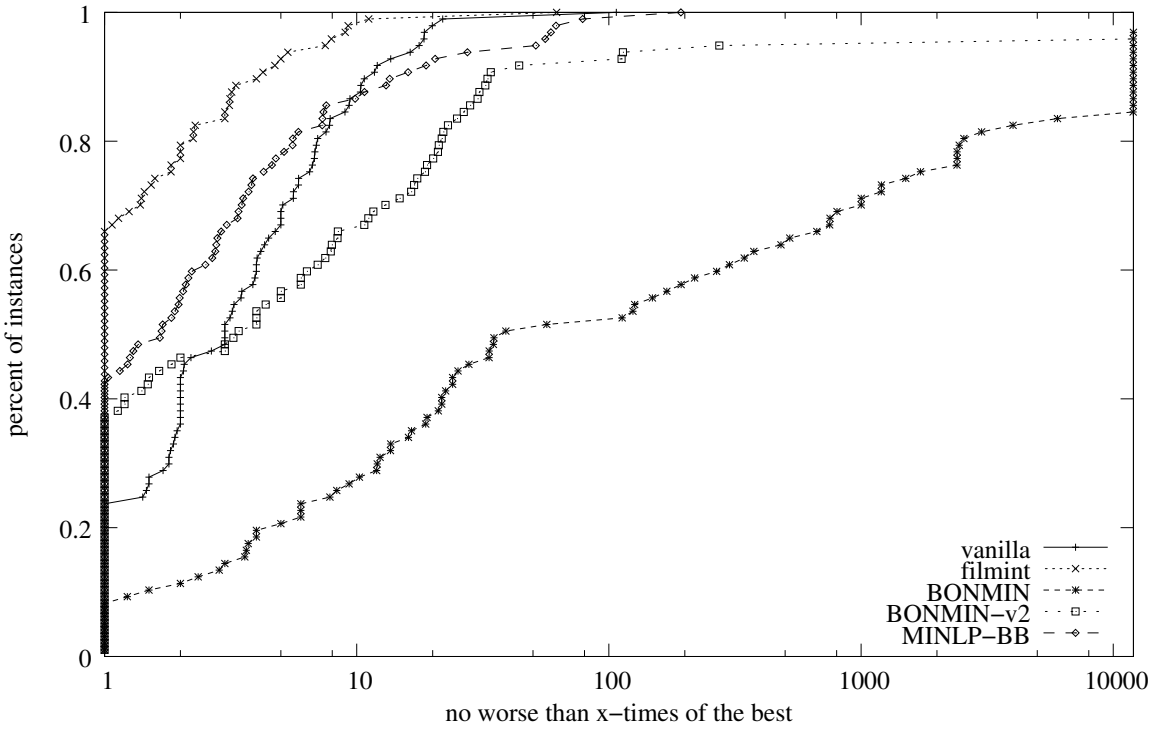

Figure 2.23: Performance profile comparing Solvers on easy instances.

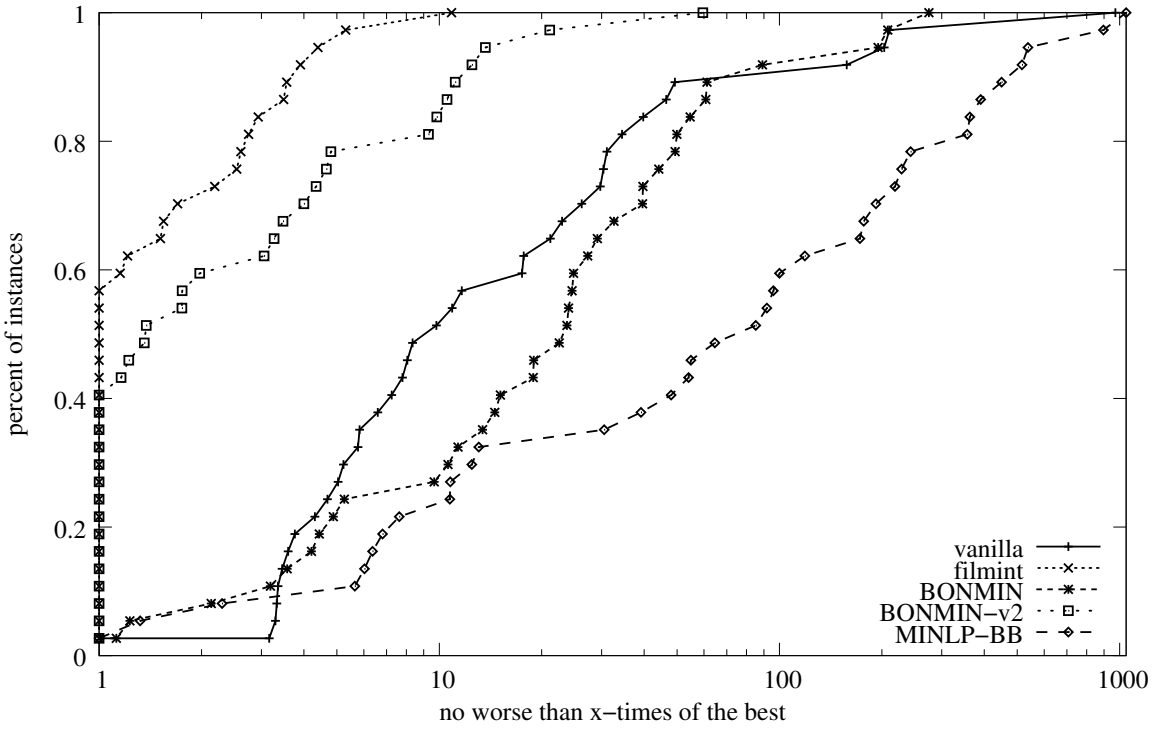

Figure 2.24: Performance profile comparing Solvers on moderate instances.

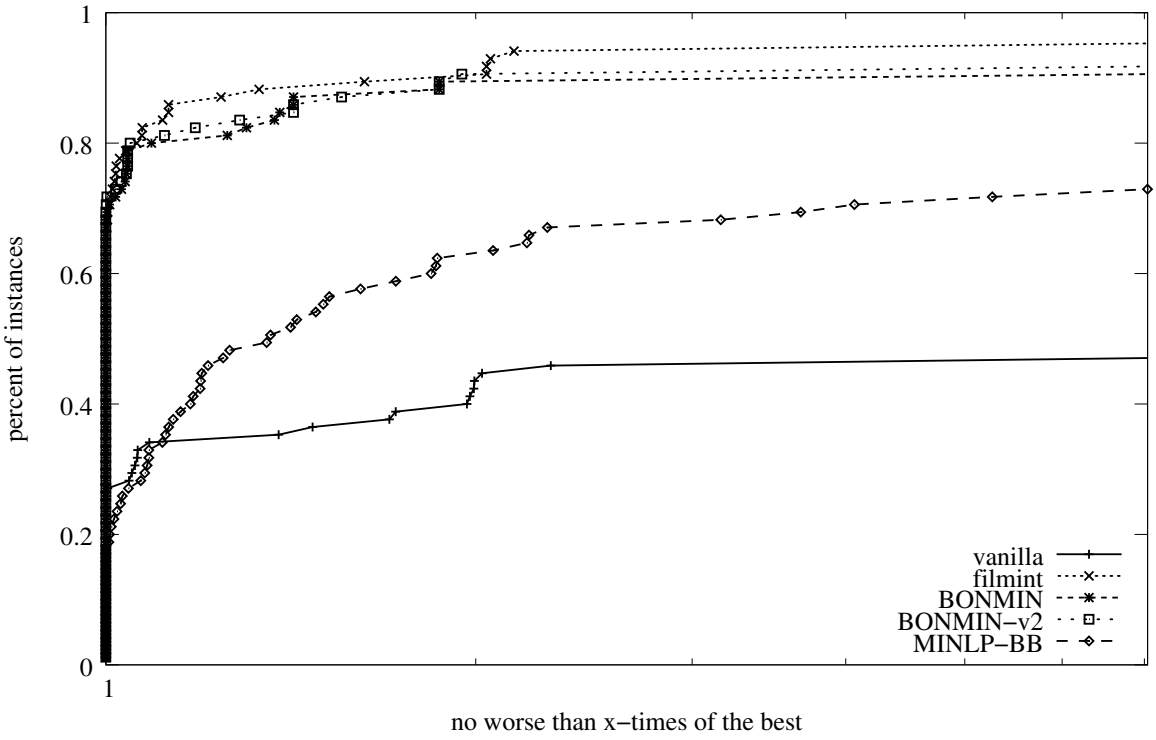

Figure 2.25: Performance profile comparing Solvers on difficult instances.

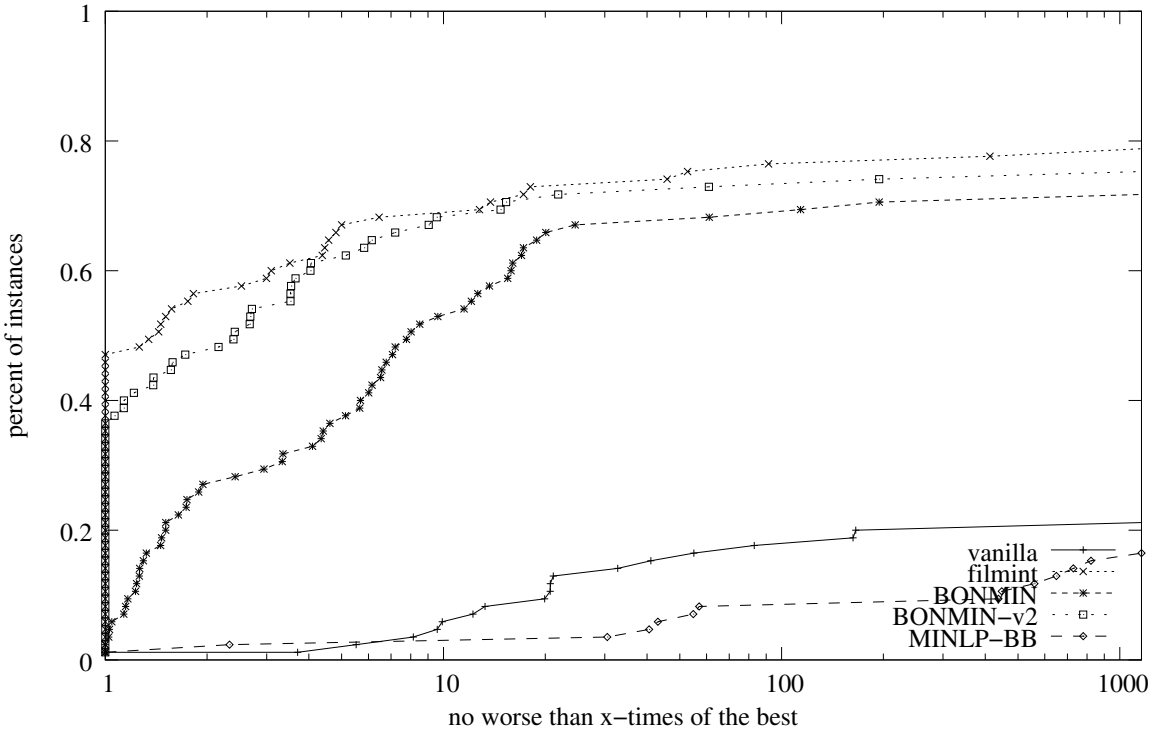

Figure 2.26: Time-based Performance profile comparing Solvers on difficult instances.

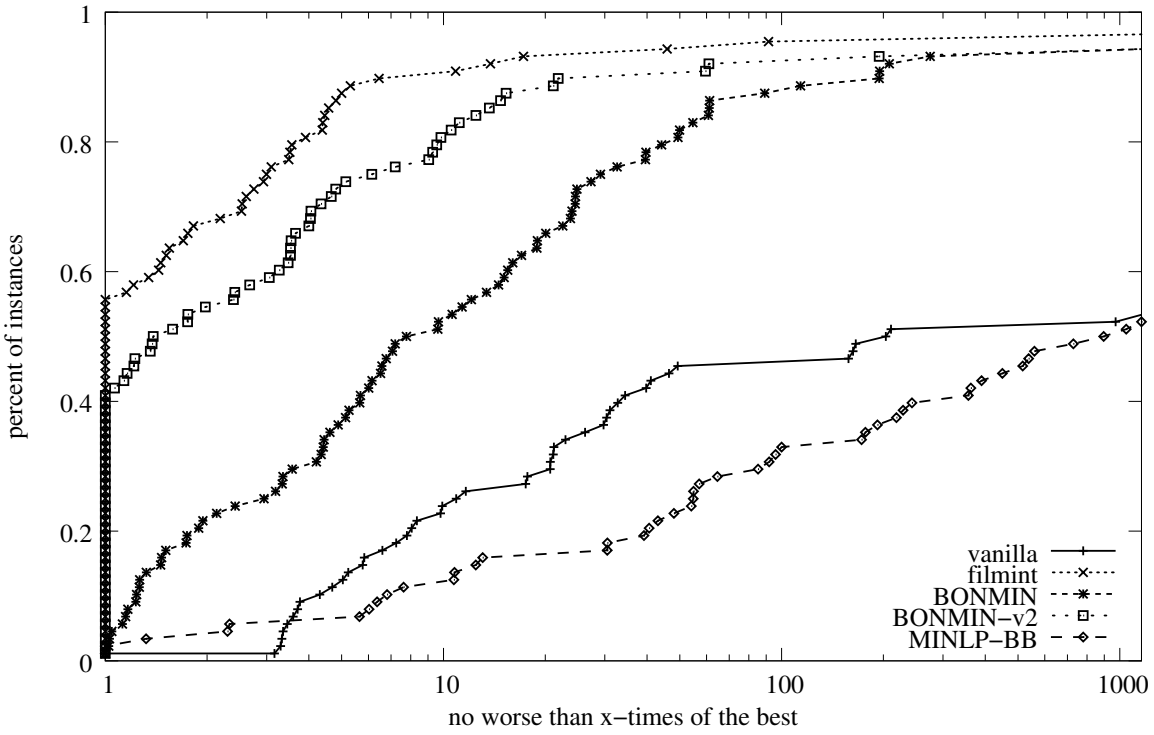

Figure 2.27: Performance Profile Comparing Solvers on Solved Instances

scheme. By using existing software components, and by enhancing the linearization procedure of the algorithm, a robust and efficient solver, competitive with all other known software available for this problem class, was created. This flexible and efficient framework provides a means to investigate new heuristic techniques and cutting planes, thereby enhancing FilMINT's effectiveness. FilMINT is available for use on the NEOS Server at `http://neos.mcs.anl.gov/neos/solvers/minco:FilMINT/AMPL.html`.

Chapter 3

Branch-and-Pump Heuristics for MINLP

Finding good integer feasible points quickly is an important aspect of algorithms for solving MINLPs. Integer feasible points provide upper bounds that can limit the size of the search tree. In addition, some problem instances cannot be solved to optimality because of the lack of good upper bounds.

In this chapter, we explore three heuristics for finding integer feasible points based on the feasibility pump algorithm that has been recently introduced for MILPs by Fischetti et al. [2005]. The first approach alternates between rounding and a nonlinear program that minimizes the distance to the nonlinear feasible set. The second approach extends the feasibility pump of [Bonami et al., 2006], and replaces the rounding step by an approximate solution of a mixed-integer linear master program. Finally, our third approach integrates the feasibility pump within an LP/NLP-based branch-and-cut framework, and avoids the re-solution of potentially expensive MILP master programs. We present detailed numerical results on a range of difficult test

problems highlighting the effectiveness of our approach.

3.1 Introduction

There exists a number of methods for finding the optimal solution of convex (MINLP). These methods can be broadly classified as those based on nonlinear programming relaxation based branch-and-bound, and those based on polyhedral relaxations. Recently, two new hybrid solvers have emerged that combine branch-and-cut with polyhedral outer-approximations including BONMIN (Bonami et al. [2008]), and FilMINT (see Chapter 2 for more details).

However, MINLPs are an NP-hard class of problems. Combinatorial explosion lies at the heart of the challenge of solving such problems, where the number of nodes increases exponentially with increases in the problem size. This is illustrated in Figure 3.1, which shows that the time taken to solve problems can increase exponentially. While state-of-the-art solvers can solve some problem classes successfully in a reasonable amount of time, other classes of problems almost completely elude our computational tools. Even worse, in many of these problems, we are unable to even detect an integer feasible solution. In this work, we develop novel heuristic techniques that aim to enlarge the class of problems for which integer feasible solutions can be found.

Primal heuristics aim to find good feasible solutions quickly. A good quality solution, if found at the beginning of the tree search procedure can significantly reduce the number of nodes enumerated and the time taken to solve the problem. Finding a feasible solution to a given MINLP is often as hard as finding an optimal solution. When compared to MILPs, this is partly due to the fact that the relaxation

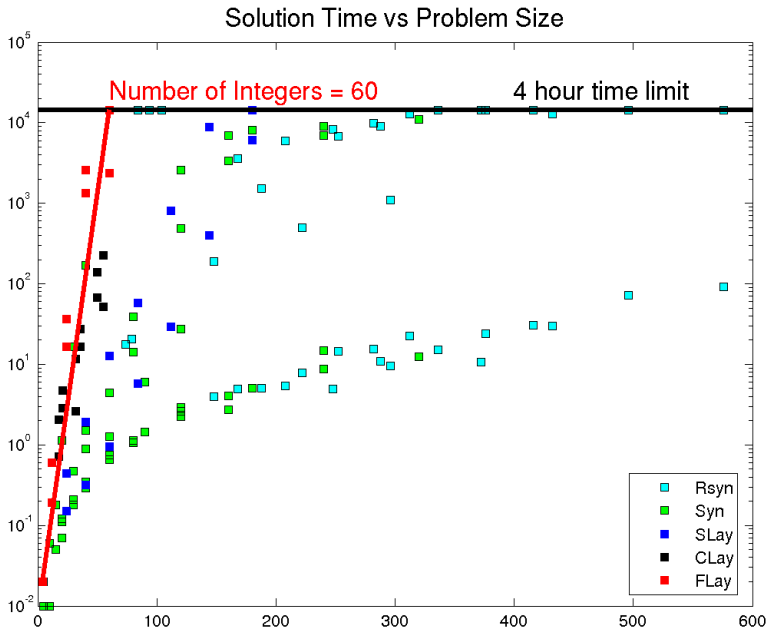

Figure 3.1: Plot showing the effect of increase in number of integer variables with time for some instances

of the MINLP obtained by using outer-approximation linearizations may be quite weak.

The feasibility pump by Fischetti et al. [2005] has been shown to be a successful heuristic for generating integer solutions for MILPs. Other successful approaches to local-search for MILPs, such as local branching (Fischetti and Lodi [2002]) and RINS (Danna et al. [2005]) can only be applied if an initial integer feasible solution is known. Bertacco et al. [2006] have extended the work of Fischetti et al. [2005] for general mixed-integer linear programs. Achterberg and Berthold [2005] have further improved the feasibility pump algorithm so as to get better integer feasible solution by using the original objective function in the search. Recently, Bonami et al. [2006] have extended the feasibility pump to MINLPs. The authors replace the rounding step with an MILP master problem that must be solved at every iteration, creating

a potential bottleneck. We will explain their approach in more detail later in this Chapter.

We explore three new approaches to obtain integer feasible solutions for MINLP. For convenience, we assume that $y \in Y \cap \{0,1\}^p$. Also, we focus our efforts on convex MINLPs. First, we develop a feasibility pump based scheme that is closely related to the pump of Fischetti et al. [2005], and which alternates between rounding and the solution of an nonlinear distance minimization problem. Next, we propose improvements to the MINLP feasibility pump of Bonami et al. [2006] by allowing incomplete MILP solutions, adding more cuts to better represent the nonlinear feasible set, and improving the scheme to get better incumbents. Finally, we propose a new branch-and-pump methodology that avoids the resolution of the MILP master problem, and instead updates the tree, as new outer approximation cuts are added to the master program. All three feasibility pumps are compared on a range of difficult MINLP test problems that show that the heuristics can find feasible solution for unsolved MINLPs, and improve the solution time of branch-and-cut solvers such as FilMINT.

The organization of this chapter is as follows: In Section 3.2, we introduce some of the subproblems that will be needed for explaining the algorithms. We explain our extension to the feasibility pump by Fischetti et al. [2005], and suggest different rounding strategies. In Section 3.3, we suggest a variant of the algorithm of Bonami et al. [2006], highlighting differences of our approach. In Section 3.4, we come up with a way to integrate the solution of NLP subproblems within the MILP-based feasibility pump framework. In Section 3.5, we present detailed computational experiments on a set of challenging convex instances, comparing the different approaches.

3.2 Rounding-Based Nonlinear Feasibility Pump

Before outlining a new nonlinear feasibility pump, we define some useful subproblems. The NLP relaxation of (MINLP) for bounds (l, u) on integer variables y is defined in Chapter 1. However, for convenience, the NLP relaxation of (MINLP) at the root node is repeated here:

$$\begin{aligned} z_{\text{NLPR}} = \text{minimize} \quad & f(x, y) \\ \text{subject to} \quad & g(x, y) \leq 0, \\ & x \in X, y \in Y. \end{aligned} \tag{NLPR}$$

The NLP subproblem for fixed values of integer decision variables y (say y^k) is defined as (NLP(y^k)) in Chapter 1. We denote the rounding of a non-integral vector y by $\hat{y} := [y]$. We discuss different strategies for rounding non-integral solutions in Section 3.2.1.

Let the set of feasible points in (NLPR) be given by

$$S := \{(x, y) : g(x, y) \leq 0,\ x \in X,\ y \in Y\} \tag{3.2.1}$$

We measure the distance to integrality between a point $(x, y) \in S$, and an integer point $(\hat{x}, \hat{y})$ by the ℓ_1- distance function defined over only the integer variables y as:

$$\Delta(y, \hat{y}) := \|y - \hat{y}\|_1. \tag{3.2.2}$$

Observe that for $y \in Y \cap \{0,1\}^p$, this expression simplifies to

$$\Delta(y, \hat{y}) = \sum_{\hat{y}_j=0} y_j + \sum_{\hat{y}_j=1} (1 - y_j).$$

Finally, given an integer point $\hat{y}^k$, the closest nonlinearly feasible point to $\hat{y}^k$ is found by solving the following NLP:

$$\begin{aligned} \text{minimize} \quad & \Delta(y, \hat{y}^k) \\ \text{subject to} \quad & g(x, y) \leq 0 \\ & x \in X,\ y \in Y. \end{aligned} \qquad \text{(NLP-}\Delta(\hat{y}^k)\text{)}$$

The basic principle of the feasibility pump (Fischetti et al. [2005]) is to generate two sequences of points. The first sequence, $(\bar{x}^0, \bar{y}^0), \ldots, (\bar{x}^k, \bar{y}^k)$ satisfies the continuous relaxation of the problem. The second sequence $(\hat{x}^1, \hat{y}^1), \ldots, (\hat{x}^{k+1}, \hat{y}^{k+1})$ satisfies the integrality requirements of the problem, but may not be feasible. This sequence of points is obtained by component-wise rounding of the points in the first sequence to the nearest integer. The authors note that this procedure may cycle and thus resort to randomization to break the cycling. Note that superscripts are used to identify the iteration number and this notation will be followed in the rest of the chapter.

We extend the feasibility pump scheme by Fischetti et al. [2005] to MINLPs by generating two sequence of points. The first sequence, $(\bar{x}^0, \bar{y}^0), \ldots, (\bar{x}^k, \bar{y}^k)$ contains points that are in the nonlinear feasible region (3.2.1) and is obtained by solving (NLP-$\Delta(\hat{y}^k)$). The second sequence $(\hat{x}^1, \hat{y}^1), \ldots, (\hat{x}^{k+1}, \hat{y}^{k+1})$ satisfies the integrality requirements ($y \in \{0,1\}^p$) and is obtained by rounding the solution of (NLP-$\Delta(\hat{y}^k)$). We employ different rounding strategies and suggest some new strategies that make

use of the nonlinear information for rounding. At any iteration k, let $(\bar{x}^k, \bar{y}^k)$ denote the solution of (NLP-$\Delta(\hat{y}^k)$). When the distance $\Delta(\bar{y}^k, \hat{y}^k)$ is zero, this implies that $(\bar{x}^k, \bar{y}^k)$ is an integer feasible solution for (MINLP). Otherwise, we obtain the next integer iterate by rounding the solution of (NLP-$\Delta(\hat{y}^k)$). Our aim is to reduce the distance between the nonlinear feasible and the integral iterates until we obtain an integer feasible solution.

We start by solving the continuous relaxation (NLPR) of (MINLP). This gives us a point $\bar{z}^0 = (\bar{x}^0, \bar{y}^0)$. This starting point is chosen with the intention of increasing the chance of finding an integer feasible solution with a good objective value. $\bar{z}^0$ is then rounded to get the point $\hat{z}^1 := (\hat{x}^1, \hat{y}^1)$. We then solve the nonlinear problem (NLP-$\Delta(\hat{y}^k)$). The solution of this nonlinear problem gives us the (possibly fractional) nonlinearly feasible point $\bar{z}^i = (\bar{x}^i, \bar{y}^i)$ where i is the iteration number. If this point is an integer point, then this is an integer feasible point for the MINLP problem and we are done. Otherwise, we round this solution using a rounding function to get $\bar{z}^{i+1} = (\bar{x}^{i+1}, \bar{y}^{i+1})$. We continue this process of solving the (NLP-$\Delta(\hat{y}^k)$) and rounding until we get an integer feasible point, or have reached the imposed time or iteration limit. The algorithm is now formally described in pseudo-code form in Algorithm 6.

Two steps in this algorithm require a more detailed explanation: The algorithm may *stall* if the rounding of the solution of (NLP-$\Delta(\hat{y}^k)$) equals the current integer iterate, $\hat{y}^{k+1} = \hat{y}^k$. If this happens, we randomly flip the T =rand$(T_0/2, 3T_0/2)$ entries $\hat{y}^k_j$ with the largest deviation $\delta_j = |\bar{y}^k_j - \hat{y}^k_j|$, where $\delta_j > \sigma_{min}$, a small threshold value needed for flipping a variable y_j. T_0 is the parameter for the mean number of variables flipped. For these components of $\hat{y}^k$, we flip up ($\hat{y}^k_j := \hat{y}^k_j + 1$) if $\bar{y}^k_j > \hat{y}^k_j$, otherwise we flip down ($\hat{y}^k_j := \hat{y}^k_j - 1$).

Set $k = 0$, solve (NLPR), and let $(\bar{x}^0, \bar{y}^0)$ be its solution.
Let $\hat{y}^{k+1} := [\bar{y}^k]$ be a rounding of $\bar{y}^k$.

while $\bar{y}^k$ *not integral* **do**
  Let $k := k + 1$.
  Solve (NLP-$\Delta(\hat{y}^k)$), let $(\bar{x}^k, \bar{y}^k)$ be its solution.
  Let $\hat{y}^{k+1} := [\bar{y}^k]$ be the rounding of $\bar{y}^k$
  if $\bar{y}^k$ *integral* **then terminate** (feasible point found)
  if $\hat{y}^{k+1} = \hat{y}^k$ **then**
    Modify $\hat{y}^{k+1}$ by *flipping* T random entries
  end
  if cycling *detected or after* K *cycles* **then**
    Modify $\hat{y}^{k+1}$ by randomly *perturbing* it
  end
end

Algorithm 6: Basic scheme for Rounding-based Nonlinear Feasibility Pump.

In addition, the algorithm may *cycle* if the integer solution repeats itself after some iterations ($\hat{y}^k := \hat{y}^s$ for some $s < k$). When this happens, the sequence of nonlinear feasible iterates $(\bar{y}^k, \ldots)$ will get repeated as well, causing no improvement in the solution scheme. To break cycling, we employ a heuristic that compares the previous H rounded values $\hat{y}^k, \hat{y}^{k-1}, \ldots, \hat{y}^{k-H}_j$ to detect cycling. If cycling is detected, the random perturbation scheme generates a uniform random value $\rho_j \in [-0.3, 0.7]$ and flips $\hat{y}^k_j$ if $|\bar{y}^k_j - \hat{y}^k_j| + \max(0, \rho_j) > P$, P being the perturbation parameter. Random perturbation is also invoked after $R = 100$ iterations so that longer cycles can be broken. Both heuristics are designed to make small random perturbations to the integer vectors. These schemes have been suggested by Fischetti et al. [2005], and have a number of parameters which affect the performance of the algorithm. We shall later present our computational findings to suggest good values for these parameters.

The algorithm is run until either the time limit, or the iteration limit, or the desired integrality gap is achieved. We also keep track of the number of times cycling is detected. If the cycling count limit is reached, we terminate the algorithm. In

addition, if the distance function value is not sufficiently decreased (say by 10%) compared to the previous iteration for $\gamma = 300$ consecutive iterations, we terminate the run.

3.2.1 Rounding strategies

An integer solution is obtained by rounding the solution of the NLP subproblem (NLP-$\Delta(\hat{y}^k)$). Rounding is done with the hope that the rounded solution $\hat{y}$ does not move too far from the feasible region of the MINLP relaxation (3.2.1). Rounding becomes a key to the performance of the feasibility pump algorithm. We therefore investigate a number of different rounding techniques to figure out strategies that may work well in general. Using the definition by Achterberg and Berthold [2005], the rounding function for integer variables $y \in Y \cap \mathbb{Z}^p$ is defined as

$$\hat{y}_j := [\bar{y}_j] := \begin{cases} \bar{y}_j & \text{if } \bar{y}_j \text{ is integral,} \\ \lfloor \bar{y}_j + \tau \rfloor & \text{otherwise.} \end{cases}$$

We first look at three different rounding strategies. The first is rounding to the nearest integer, for which $\tau := 0.5$. The second one is random quadratic rounding, for which a random number $\beta \in [0, 1]$ is generated. The parameter τ for random quadratic rounding is then defined to be

$$\tau := \begin{cases} 2\beta(1-\beta) & \beta \leq 0.5, \\ 1 - 2\beta(1-\beta) & \text{otherwise.} \end{cases}$$

This is illustrated in Figure 3.2. The third strategy, called random threshold based rounding, generates a random number $\beta \in [0, 1]$, and updates τ as $\tau := \beta/2 + 0.25$. For MILPs, Achterberg and Berthold [2005] report that random quadratic rounding works best. Later in Section 3.5, we present computational results for showing the effect of rounding strategies.

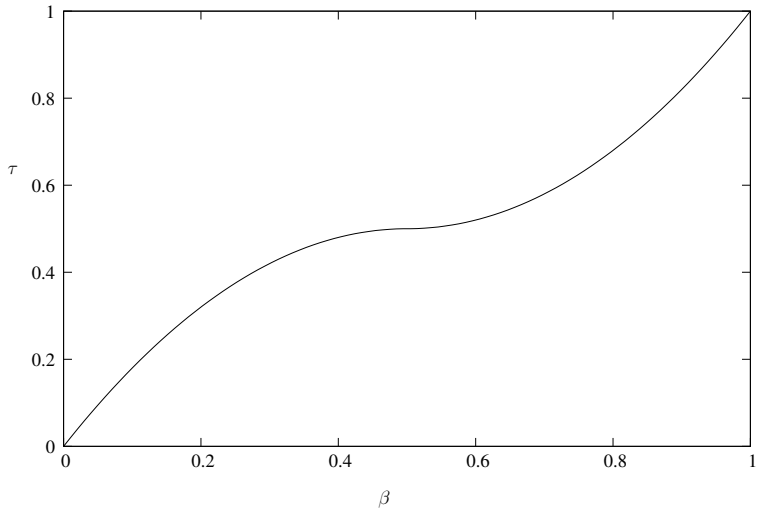

Figure 3.2: Plot of τ vs β for random quadratic rounding.

Many problems contain special ordered sets of type one, e.g. to model discrete choices such as $w \in \{d_1, \ldots, d_k\}$, i.e.

$$1 = \sum_{i=1}^{k} y_i, \quad \text{and} \quad w = \sum_{i=1}^{k} d_i y_i.$$

We note, that rounding individual binary variables could violate the SOS condition. For example, if $y^* = (1/k, \ldots, 1/k)$, then $\hat{y} = 0$ is the rounding. It is better to round on the special ordered set, i.e. find the nearest d_i to the relaxation w, and then round the SOS accordingly. To do this, we calculate a score s_j for this SOS-1 set as $s_j := \sum_{i=1}^{k} j y_j$. We then use one of the three rounding schemes (simple, quadratic, or random threshold) for rounding s_j. Our scheme for SOS-1 based rounding then

sets the variables y_j in the set as

$$\hat{y}_j := \begin{cases} 1 & j = [s_j], \\ 0 & \text{otherwise.} \end{cases} \tag{3.2.3}$$

This scheme preserves the SOS condition and depending on the weights, the SOS variable can now take any of the possible values in the set $\{d_1, \ldots, d_k\}$. Furthermore, in case of flipping, we use s_j for flipping instead of each of the variables y_j in the set.

3.2.2 Handling General Integer Variables

The nonlinear feasibility pump is readily extended to MINLPs with general integer variables ($y_j \in [l_j, u_j]$) by following the same approach as that of Bertacco et al. [2006]. Following the definition of the distance function (3.2.2), the objective function of the minimum-distance problem, (NLP-$\Delta(\hat{y}^k)$), becomes

$$\Delta(y, \hat{y}^k) := \sum_{\hat{y}_j^k = l_j} (y_j - l_j) + \sum_{\hat{y}_j^k = u_j} (u_j - y_j) + \sum_{l_j < \hat{y}_j^k < u_j} d_j,$$

where $d_j = \|y_j - \hat{y}_j\|_1$, and l_j and u_j are the lower and upper bounds on integer variables y_j. The nonsmooth constraint $d_j = \|y_j - \hat{y}_j\|_1$ can be reformulated equivalently as

$$d_j \geq y_j - \hat{y}_j$$

$$d_j \geq \hat{y}_j - y_j.$$

Thus, for every general integer variable not at its lower or upper bound, we need to add one variable and two constraints to (NLP-$\Delta(\hat{y}^k)$). Bertacco et al. [2006] handle the case of general integer variables by running the feasibility pump in three stages. We run the nonlinear feasibility pump for MINLPs in a similar way in 2 stages. In the first stage, the general integer variables are treated as continuous variables, and the pumping is done only with respect to the binary variables in the problem. This helps us to quickly get a solution which is feasible with respect to binary variables, with the hope that most of the integer variables lie at their upper or lower bounds at the optimal solution of (NLP-$\Delta(\hat{y}^k)$). On achieving a solution like this, we stop the first stage and use this solution as the starting solution for the second stage. In this stage, the integrality of general integer variables is restored. Pumping is then done on all binary and general integer variables. The use of the first stage solution helps to possibly reduce the number of new variables and constraints. We continue with the iterative procedure of solving (NLP-$\Delta(\hat{y}^k)$) and rounding until some termination criteria is satisfied.

The framework allows for using different parameter values for the second stage compared to the first. Our computational experiments attempt to obtain good parameter values for each of the stages, which work well for MINLP instances in general.

3.2.3 Improving the Quality of the Incumbent

In general, the feasibility pump procedure may not provide a solution with a sufficiently low objective value, because the algorithm ignores $f(x, y)$ when solving (NLP-$\Delta(\hat{y}^k)$). Here, we discuss strategies to improve an incumbent integer feasible point.

Let $(\bar{x}^*, \bar{y}^*)$ be an integer feasible solution obtained by running the feasibility pump algorithm. Let $\hat{z}^* := f(\bar{x}^*, \bar{y}^*)$ be the solution value corresponding to this incumbent.

The pump can be used to improve the quality of the solution by running them again with an additional constraint on (NLP-$\Delta(\hat{y}^k)$):

$$f(x, y) \leq UB, \tag{3.2.4}$$

where the upper bound UB is defined as $UB := \alpha z_{\text{NLPR}} + (1 - \alpha)\hat{z}^*$ for some small $\alpha \in [0, 1]$ (say 0.05).

It is to be noted that an incumbent $(\bar{x}^*, \bar{y}^*)$ may not be the best solution (with respect to the objective function $f(x, y)$) for the integer assignment $\bar{y}^*$. We improve this solution by solving (NLP(y^k)) for the integer assignment $\bar{y}^*$. The optimal solution of (NLP(y^k)) is the best possible solution that can be obtained for a fixed integer assignment $\bar{y}^*$. This scheme also helps reduce the number of iterations, because the best solution for a given $\bar{y}^*$ is obtained in one iteration.

3.3 A modified MILP-based Feasibility Pump

The rounding-based pump presented in the previous section can be handy in getting integer feasible solutions to (MINLP) quite quickly. However, it can also fail to obtain integer feasible solutions for some instances due to cycling and stalling. In this section, we outline two variants of the feasibility pump algorithm that solve MILPs in order to get an integer solution rather than round.

We start by describing the feasibility pump scheme of Bonami et al. [2006], that alternatively solves a distance-based NLP and a distance-based MILP to optimality. We then suggest some improvements to the algorithmic scheme that aim at reducing the time spent on solving MILP, and reducing the number of iterations taken in finding

integer feasible solutions. We also describe how to obtain a better approximation of the feasible region and the objective function in these methods.

3.3.1 The Feasibility Pump of Bonami et al. [2006]

The feasibility pump algorithm by Bonami et al. [2006] constructs the sequence of integer points $(\hat{x}^1, \hat{y}^1), \ldots, (\hat{x}^{k+1}, \hat{y}^{k+1})$ by solving a MILP that outer-approximates the feasible region of (MINLP). Convexity of nonlinear functions ($g(x, y) \leq 0$) imply that linearizations about any feasible solution (x^k, y^k) form an outer approximation of the feasible region (3.2.1). The outer-approximations are expressed as:

$$g(x^k, y^k) + \nabla g(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0, \qquad (\text{OAC}(x^k, y^k))$$

where $\nabla g(x^k, y^k)^T = [\nabla g_1(x^k, y^k) : \ldots : \nabla g_m(x^k, y^k)]^T$ is the Jacobian of m nonlinear constraints.

Given a set of points $\mathcal{K} = \{(x^0, y^0), \ldots, (x^k, y^k)\}$, a relaxation of the feasible set of (MINLP) using inequalities (OAC(x^k, y^k)) is:

$$\mathcal{OAC}(\mathcal{K}) = \{(x, y) \in X \times Y \cap \mathbb{Z}^p \mid (x, y) \text{ satisfy OAC}(x^k, y^k)\ \forall (x^k, y^k) \in \mathcal{K}\}.$$

The closest integer point to $(\bar{x}^{i-1}, \bar{y}^{i-1})$ at iteration i is then found by solving the distance-based MILP ((FOA)^i).

$$\text{minimize} \quad \|y - \bar{y}^{i-1}\|_1$$

$$\text{subject to} \quad g(x^k, y^k) + \nabla g(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0, \quad \forall k = 0, \ldots, i-1 \quad ((\text{FOA})^i)$$

$$x \in X, y \in Y \cap \mathbb{Z}^p.$$

The algorithm starts by solving (NLPR) to get an initial point $(\bar{x}^0, \bar{y}^0)$. At iteration i, an integer point $(\hat{x}^i, \hat{y}^i)$ is found by solving the MILP $((\text{FOA})^i)$ to optimality. The closest nonlinear feasible point $(\bar{x}^i, \bar{y}^i)$ is then found by solving the distance-based NLP $((\text{FP-NLP})^i)$.

$$\begin{aligned} \text{minimize} \quad & \|y - \hat{y}^i\|_2 \\ \text{subject to} \quad & g(x, y) \leq 0 \\ & x \in X, \ y \in Y. \end{aligned} \qquad ((\text{FP-NLP})^i)$$

The basic version of the algorithm of Bonami et al. [2006] iteratively solves $((\text{FOA})^i)$ and $((\text{FP-NLP})^i)$. The authors prove that this scheme does not cycle as long as linear independence constraint qualification (LICQ) holds. The enhanced version of the feasibility pump prevents cycling by adding the inequality

$$(\bar{y}^k - \hat{y}^k)^T (y - \bar{y}^k) \geq 0. \tag{3.3.5}$$

to $((\text{FOA})^i)$ for every iteration $i > k$. This cuts off the integer point $(\hat{x}^k, \hat{y}^k)$, obtained by solving $((\text{FOA})^i)$ at iteration k, that lies outside the feasible region (3.2.1).

When a feasible solution $(\hat{x}^k, \hat{y}^k)$ (with objective value z_{UB}) to (MINLP) is found,

the feasibility pump scheme attempts to improve the incumbent by adding the linearization (3.3.6) of the objective function to the MILP ((FOA)^i).

$$\eta \geq f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix}. \tag{3.3.6}$$

The upper bound on the variable $\eta \in \mathbb{R}^1$ is set to $z_{UB} - \delta$ for a small $\delta > 0$ each time a new incumbent is found. This linearization cuts off the current incumbent $(\hat{x}^k, \hat{y}^k)$. The iterative scheme is then restarted about the solution $(\bar{x}^0, \bar{y}^0)$ of the relaxation (NLPR).

The authors prove that the MILP-based feasibility pump does not cycle. This implies that the algorithm takes a finite number of iterations for finding integer feasible solutions when the integer variables are bounded. Also, this algorithm is an exact solution scheme for convex (MINLP), either an integer feasible solution is found or infeasibility of (MINLP) is proved.

3.3.2 Modifying the MILP-based Feasibility Pump

The MILP-based feasibility pump by Bonami et al. [2006] has a number of potential shortcomings: It requires the solution of an MILP at every iteration, which could become a computational bottleneck. Also, their feasibility pump uses an ℓ_2-distance problem, rather than an ℓ_1-distance problem that is more likely to produce a solution that is a vertex, and therefore integer.

Our modified scheme follows the same principle as the algorithm of Bonami et al. [2006] in that it iteratively solves distance-based NLPs and MILPs. However, in addition, we propose three modifications to improve the MILP-based feasibility pump:

1 We have observed that an integer feasible solution to ((FOA)^i) is usually found quickly, but that the branch-and-cut scheme takes much longer to prove optimality. While solving ((FOA)^i) to optimality gives us the closest integer point from a point $(\bar{x}^k, \bar{y}^k)$ in the feasible region (3.2.1), this usually hinders the process of getting integer feasible solutions quickly. It is to be noted that the proof by Bonami et al. [2006] on cycling does not rely on the optimality of the solution of ((FOA)^i). In fact, any integer solution to ((FOA)^i) can be used for setting up and solving a distance-based NLP. A more flexible approach is to stop the tree search as soon as the r^{th} integer solution is found for some $r \geq 1$. In particular, we suggest and experiment with stopping the tree search for ((FOA)^i) as soon as the first integer solution is found. This modification is reminiscent of a suggestion to speed up outer approximation by Duran and Grossmann [1986].

2 The first integer solution, say $(\hat{x}^k, \hat{y}^k)$, is likely to lie outside the feasible region (3.2.1). In this case, the algorithm may take more iterations to reach an integer feasible point. We suggest to add cuts to produce a tighter relaxation of the feasible region (3.2.1). Any integer point $(\hat{x}^k, \hat{y}^k)$ lying outside (3.2.1), can be cut off by using linearizations (OAC(x^k, y^k)) about that point. Please see the result by Fletcher and Leyffer [1994] in this regard.

3 In case LICQ does not hold, the linearizations may not cut off the integer point that lies outside the nonlinear feasible region. Thus, one needs to add the inequality (3.3.5) to the MILP to cut off the infeasible integer point $(\hat{x}^k, \hat{y}^k)$. By using the ℓ_1-distance minimization, the inequality (3.3.5) is no longer valid. Instead, we obtain a inequality that cuts off $(\hat{x}^k, \hat{y}^k)$, where $\bar{y}^k$ is obtained as

the solution of (NLP-$\Delta(\hat{y}^k)$) from

$$\sum_{j:\ \hat{y}_j^k = l_j} (y_j - \bar{y}_j^k) - \sum_{j:\ \hat{y}_j^k = u_j} (y_j - \bar{y}_j^k) \geq 0. \tag{3.3.7}$$

This inequality is obtained by applying the results by Bonami et al. [2006] for the ℓ_1-distance function.

We can now state the MILP master that is solved at every iteration of the modified feasibility pump. Let $\mathcal{I} := \mathcal{I}_f \cup \mathcal{I}_d$ be the set of integer points obtained by solving MILPs iteratively. Here, $\mathcal{I}_f$ is the set of points that are integer feasible and $\mathcal{I}_d$ is the set of integer points that lie outside the feasible region (3.2.1). Also, let $\mathcal{C}$ be a collection of points obtained by solving (NLP-$\Delta(\hat{y}^k)$) at various integer solutions $\hat{y}^k$ during the algorithm. The distance-based MILP constructed using points $\mathcal{K} := \mathcal{C} \cup \mathcal{I}$ at iteration i is now defined as (MP$\Delta(\mathcal{K})^i$).

$$\begin{aligned}
\text{minimize} \quad & \|y - \bar{y}^{i-1}\|_1 \\
\text{subject to} \quad & g(\bar{x}^k, \bar{y}^k) + \nabla g(\bar{x}^k, \bar{y}^k)^T \begin{bmatrix} x - \bar{x}^k \\ y - \bar{y}^k \end{bmatrix} \leq 0, \quad \forall (\bar{x}^k, \bar{y}^k) \in \mathcal{C} \qquad (\text{MP}\Delta(\mathcal{K})^i) \\
& g(\hat{x}^k, \hat{y}^k) + \nabla g(\hat{x}^k, \hat{y}^k)^T \begin{bmatrix} x - \hat{x}^k \\ y - \hat{y}^k \end{bmatrix} \leq 0, \quad \forall (\hat{x}^k, \hat{y}^k) \in \mathcal{I}_d \\
& \sum_{j:\ \hat{y}_j^k = l_j} (y_j - \bar{y}_j^k) - \sum_{j:\ \hat{y}_j^k = u_j} (y_j - \bar{y}_j^k) \geq 0, \quad \forall (\hat{x}^k, \hat{y}^k) \in \mathcal{I}_d \\
& x \in X, y \in Y \cap \mathbb{Z}^p.
\end{aligned}$$

3.3.3 Improving the Incumbent

A feasible solution (x^*, y^*) for (MINLP) is obtained when the objective value ($\Delta(y^*, \hat{y}^k)$) for the distance-based NLP (NLP-$\Delta(\hat{y}^k)$) becomes zero. The objective value of the solution is $f(x^*, y^*)$. However, this solution may not be the best feasible solution for the integer assignment y^*, especially since the true objective function $f(x, y)$ is not taken into account during the scheme of solving MILPs and NLPs iteratively. Given an integer assignment y^*, the solution with the lowest objective value can be found by solving the NLP subproblem (NLP(y^k)) obtained by fixing $y = y^*$ and solving the subproblem in the reduced space of x-variables.

Since the algorithm of Bonami et al. [2006] just evaluates the objective function at (x^*, y^*), this can sometimes cause the scheme to solve a large number of MILPs and NLPs. In each such iteration, the scheme may produce a different x^* for the same integer assignment y^*.

To avoid this, we solve (NLP(y^k)) whenever the distance $\Delta(y^*, \hat{y}^k)$ becomes zero. Solving (NLP(y^k)) and using its solution can significantly reduce the number of redundant iterations and speed up the solution scheme. Solving (NLP(y^k)) to obtain an integer feasible solution is similar to the procedure used in the outer-approximation algorithm (Duran and Grossmann [1986]) and the LP/NLP algorithm of Quesada and Grossmann [1992].

When an incumbent (x^*, y^*) with objective value z_{UB} is found, the algorithm of Bonami et al. [2006] add an underestimator for the objective $f(x, y)$ to the MILP ((FOA)^i) from the next iteration. The linearization (3.3.6) along with bounds on the new variable $\eta \leq z_{UB} - \delta$, for a small $\delta > 0.0$ (say 0.01), cuts off the current integer feasible solution. The bounds on η is updated whenever a new incumbent

is found. However, adding the linearization and the upper bound $z_{UB} - \delta$ does not guarantee that the next incumbent (if any) will have an objective value strictly less than z_{UB}. This is because the linearization underestimates the true objective function, and when the underestimator is poor, it can accept points for which $\eta \leq z_{UB} - \delta$ but $f(x, y) \geq z_{UB}$. This is illustrated in Figure 3.3.

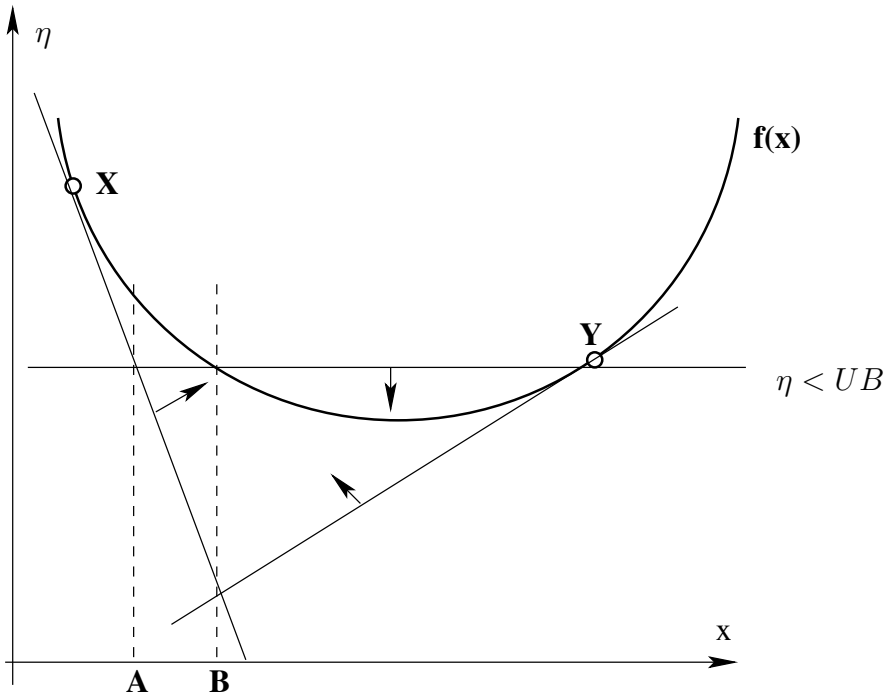

Figure 3.3: Illustration of how underestimators can fail to give strictly improving solutions. For $x \in [A, B]$, $f(x) > UB$

Let z_{UB} be the objective value corresponding to the current best integer feasible solution. We add objective linearizations (3.3.6) about every feasible point (x^k, y^k) so that we get a better approximation of the true objective function. The upper bound on the variable η is changed only when a new incumbent with strictly better objective value is found. It can easily be seen that the linearization (3.3.6) about the point (x^k, y^k), along with the upper bound on η cuts off the integer point (x^k, y^k). Thus, an integer solution that was obtained as a solution to (MP$\Delta(\mathcal{K})^i$) will not be repeated

in this scheme.

Further, each time a new incumbent with strictly better objective value is found, we reduce the upper bound on η by using a convex combination of the solution of (NLPR) and the current best solution z_{UB}. The new desired upper bound UBD is given as

$$UBD := \alpha z_{\text{NLPR}} + (1-\alpha) z_{UB}.$$

The MILP master problem can now be expressed as (OMP$\Delta(\mathcal{K})^i$).

$$\begin{aligned}
\text{minimize} \quad & \|y - \bar{y}^{i-1}\|_1 && (\text{OMP}\Delta(\mathcal{K})^i)\\
\text{subject to} \quad & g(\bar{x}^k, \bar{y}^k) + \nabla g(\bar{x}^k, \bar{y}^k)^T \begin{bmatrix} x - \bar{x}^k \\ y - \bar{y}^k \end{bmatrix} \leq 0, \quad \forall (\bar{x}^k, \bar{y}^k) \in C \\
& g(\hat{x}^k, \hat{y}^k) + \nabla g(\hat{x}^k, \hat{y}^k)^T \begin{bmatrix} x - \hat{x}^k \\ y - \hat{y}^k \end{bmatrix} \leq 0, \quad \forall (\hat{x}^k, \hat{y}^k) \in \mathcal{I}_d \\
& \sum_{j:\, \hat{y}^k_j = l_j} (y_j - \bar{y}^k_j) - \sum_{j:\, \hat{y}^k_j = u_j} (y_j - \bar{y}^k_j) \geq 0, \quad \forall (\hat{x}^k, \hat{y}^k) \in \mathcal{I}_d \\
& \eta \geq f(\hat{x}^k, \hat{y}^k) + \nabla f(\hat{x}^k, \hat{y}^k)^T \begin{bmatrix} x - \hat{x}^k \\ y - \hat{y}^k \end{bmatrix}, \quad \forall (\hat{x}^k, \hat{y}^k) \in \mathcal{I}_f \\
& z_{\text{NLPR}} \leq \eta \leq UBD \\
& x \in X, y \in Y \cap \mathbb{Z}^p.
\end{aligned}$$

When an incumbent is found, one can reset the objective function of the MILP using the solution of (NLPR). By doing this, the branch-and-cut procedure aims to find a feasible solution to (OMP$\Delta(\mathcal{K})^i$) by minimizing the ℓ_1-distance about $(\bar{x}^0, \bar{y}^0)$.

However, we note, that using a different point, say the current incumbent $(\bar{x}^*, \bar{y}^*)$, can lead to the diversification of the search. We use this strategy in our implementation.

The proposed algorithm iteratively solves (OMP$\Delta(\mathcal{K})^i$) and (NLP-$\Delta(\hat{y}^k)$) until either (OMP$\Delta(\mathcal{K})^i$) becomes infeasible or a time limit or a desired integrality gap is reached. The modified MILP-based feasibility pump algorithm is now given in pseudo-code form in Algorithm 7. The index k is used to count the iteration number.

Set $k = 0$, $\mathcal{C} := \emptyset$, $\mathcal{I}_d := \emptyset$, $\mathcal{I}_f := \emptyset$, $UB := \infty$, $UBD := \infty$
Solve (NLPR), solution is $(\bar{x}^k, \bar{y}^k)$ with objective z_{NLPR}
$\mathcal{C} := \mathcal{C} \cup \{(\bar{x}^k, \bar{y}^k)\}$
while *termination criteria* not *satisfied* **do**
  Let $k := k + 1$. $i := k$ $\mathcal{K} := \mathcal{C} \cup \mathcal{I}_f \cup \mathcal{I}_d$
  Use $\bar{y}^{k-1}$ to set the objective of (OMP$\Delta(\mathcal{K})^i$)
  Solve (OMP$\Delta(\mathcal{K})^i$) until *first* integer solution $(\hat{x}^k, \hat{y}^k)$ found.
  if *(OMP*$\Delta(\mathcal{K})^i$*)* infeasible **then terminate**
  Use $\hat{y}^k$ to set the objective of (NLP-$\Delta(\hat{y}^k)$)
  Solve (NLP-$\Delta(\hat{y}^k)$). Solution is $(\bar{x}^k, \bar{y}^k)$ with objective value Δ
  if $\Delta = 0$ **then**
    Solve (NLP(y^k)). Solution is $(\bar{x}^k, \bar{y}^k)$ with objective $z_{\text{NLP}(y^k)}$
    $UB := Min\{UB, z_{\text{NLP}(y^k)}\}$, $UBD := \alpha z_{\text{NLPR}} + (1 - \alpha)UB$
    $\mathcal{C} := \mathcal{C} \cup \{(\bar{x}^k, \bar{y}^k)\}$, $\mathcal{I}_f := \mathcal{I}_f \cup \{(\bar{x}^k, \bar{y}^k)\}$
  else
    $\mathcal{C} := \mathcal{C} \cup \{(\bar{x}^k, \bar{y}^k)\}$, $\mathcal{I}_d := \mathcal{I}_d \cup \{(\hat{x}^k, \hat{y}^k)\}$
  end
end

Algorithm 7: Modified MILP-based Nonlinear Feasibility Pump

The proposed MILP-based feasibility pump scheme has the same theoretical properties as the algorithm of Bonami et al. [2006]. The algorithm does not cycle, takes a finite number of iterations and is an exact solution methodology for convex (MINLP). This is because of the fact that the proof of cycling does not depend on the quality of the integer solution. Further, the linearizations and the constraint (3.3.7) are valid for the feasible region of (MINLP). Our computational runs, presented in section (3.5) shows the effectiveness of this solution procedure for getting integer feasible

solutions.

3.4 Branch-and-Pump Heuristic

In this section we outline another feasibility-pump-based algorithm, which we call *branch-and-pump*. This pump is related to the feasibility pump of Bonami et al. [2006], but avoids the re-solution of the MILP master problem, similar to the way in which the LP/NLP branch-and-bound algorithm of Quesada and Grossmann [1992] avoids the re-solution of the MILP master problem.

The feasibility pump of Bonami et al. [2006] follows the philosophy of outer approximation (Duran and Grossmann [1986]) by solving MILPs and NLPs iteratively. We note that solving MILPs iteratively may turn out to be a bottleneck in the solution procedure for obtaining feasible solutions.

Instead, the branch-and-pump algorithm follows an LP/NLP-like approach. In this approach, the tree search of the MILP (OMP$\Delta(\mathcal{K})^i$) is interrupted whenever an integer solution is found. A distance-based NLP (NLP-$\Delta(\hat{y}^k)$) is then solved and linearizations about the solution added to (OMP$\Delta(\mathcal{K})^i$). The tree search for a new integer solution is then continued until an integer feasible solution for (MINLP) is found. This way, we only need to create and search a single branch-and-cut tree. Solving the distance-based NLP (NLP-$\Delta(\hat{y}^k)$) attempts to "pump" the integrality of an integer iterate $(\hat{x}, \hat{y})$ into the nonlinear feasible iterate $(\bar{x}, \bar{y})$. The solution also gives us linearizations that need to be added to (OMP$\Delta(\mathcal{K})^i$) to avoid cycling. We note that this can be done easily within the same search tree, without having to restart the search from scratch. Our proposed method can result in significant improvements in the solution scheme. The algorithm is called *branch-and-pump* since it integrates

"pumping" (to get a nonlinear feasible point close to an integer feasible solution) with branching (for getting integer feasible solutions) within a single algorithmic framework.

The branch-and-pump scheme starts by solving the NLP relaxation (NLPR) to get a starting solution $(\bar{x}^0, \bar{y}^0)$. This solution is used to set the linearizations and the objective for the master MILP (OMP$\Delta(\mathcal{K})^i$). We then start a branch-and-cut tree search for the MILP. Whenever an integer solution $(\hat{x}, \hat{y})$ is found, we solve the distance-based NLP (NLP-$\Delta(\hat{y}^k)$). If the solution of (NLP-$\Delta(\hat{y}^k)$), say $(\bar{x}^k, \bar{y}^k)$, is not integral, we add linearizations about the MILP and the NLP solution and continue with the tree search. We note that each time (NLP-$\Delta(\hat{y}^k)$) is solved, we should ideally change the objective of the MILP (OMP$\Delta(\mathcal{K})^i$) to $\|y - \bar{y}^k\|_1$ so that the tree now looks for the integer feasible point closest to $\bar{y}^k$. This fact makes it difficult to define an exact LP/NLP-like procedure. However, simply changing the objective during the course of the search, will not necessarily find such a point. While one can use this new objective during the same tree search as a heuristic procedure, a number of difficult issues need to be handled. The lower bounds on each of the open nodes need to be changed to take into account the new objective. Further, fathoming rules for this branch-and-cut procedure need to be re-visited. In the branch-and-cut procedure, a node can get fathomed if its lower bound is greater than the current integer feasible solution. However, such nodes may be feasible if the bounds due to the new objective change. As far as we know, there is no direct analytic relation between the solution of a node based on the old objective and the new objective.

We handle this situation by continuing to solve the MILP (OMP$\Delta(\mathcal{K})^i$) about the same objective as before. In this case, the bounds remain valid and the tree search can be continued. We note that the purpose of the MILP is to get an integer

solution that lies within the relaxation defined by the outer-approximation. While we may potentially lose on some advantages of using a different objective for the MILP, we may still gain from the fact that the MILP does not need to be solved again from scratch at each iteration. If the solution of (NLP-$\Delta(\hat{y}^k)$) provides us with an integer feasible solution, we solve (NLP(y^k)) to get the best possible incumbent for the current integer assignment $\bar{y}^k$. The branch-and-pump scheme thus involves solving (OMP$\Delta(\mathcal{K})^i$) with (NLP-$\Delta(\hat{y}^k)$) and (NLP(y^k)) solved during the tree search of (OMP$\Delta(\mathcal{K})^i$). This is continued until we get an integer feasible solution or the MILP becomes infeasible. For practical purpose, we put a time limit on the MILP run. However, we note that the objective value of (NLP-$\Delta(\hat{y}^k)$), Δ, being zero does not imply that we have a better incumbent. In this case, we can either restart the MILP as described in the next paragraph, or eliminate the current integer solution and continue with the tree search. We feel that restarting the MILP may be the better in order to diversify the search.

We can improve on the current integer feasible solution by restarting (OMP$\Delta(\mathcal{K})^i$) from scratch with the objective set about the current incumbent. We also add objective linearizations (3.3.6) and update the bounds on the variable η. This procedure is the same as described in Section 3.3.

We now formally present the pseudo-code for the branch-and-pump procedure in Algorithm 8. The proposed algorithm again shares the same properties as the algorithm of Bonami et al. [2006]. Namely, this algorithm does not cycle and is an exact solution procedure for convex MINLPs. Also, if the integer variables are bounded, this algorithm takes a finite number of iterations.

In the next section, we describe the computational setup that is used to test the performance of the proposed schemes, and present detailed computational results.

Set $k = 0$, $\mathcal{C} := \emptyset$, $\mathcal{I}_d := \emptyset$, $\mathcal{I}_f := \emptyset$, $\mathcal{K} := \emptyset$, $UB := \infty$, $UBD := \infty$
Solve (NLPR), solution is $(\bar{x}^k, \bar{y}^k)$ with objective z_{NLPR}
$\mathcal{C} := \mathcal{C} \cup \{(\bar{x}^k, \bar{y}^k)\}$

while *termination criteria* not *satisfied* **do**
  Let $k := k + 1$. $i := k$, $\mathcal{K} := \mathcal{C} \cup \mathcal{I}_f \cup \mathcal{I}_d$
  Use $\bar{y}^{k-1}$ to set the objective of (OMP$\Delta(\mathcal{K})^i$)
  Start solving (OMP$\Delta(\mathcal{K})^i$).
  while *(OMP*$\Delta(\mathcal{K})^i$*)* not *terminated* **do**
    if *integer solution* $(\hat{x}^k, \hat{y}^k)$ *found* **then**
      Use $\hat{y}^k$ to set the objective of (NLP-$\Delta(\hat{y}^k)$)
      Solve (NLP-$\Delta(\hat{y}^k)$). Solution is $(\bar{x}^k, \bar{y}^k)$ with objective value Δ
      if $\Delta = 0$ **then**
        Solve (NLP(y^k)). Solution is $(\bar{x}^k, \bar{y}^k)$ with objective $z_{\text{NLP}(y^k)}$
        $UB := Min\{UB, z_{\text{NLP}(y^k)}\}$, $UBD := \alpha z_{\text{NLPR}} + (1 - \alpha)UB$
        $\mathcal{C} := \mathcal{C} \cup \{(\bar{x}^k, \bar{y}^k)\}$, $\mathcal{I}_f := \mathcal{I}_f \cup \{(\bar{x}^k, \bar{y}^k)\}$
        Stop (OMP$\Delta(\mathcal{K})^i$) solve.
      else
        $\mathcal{C} := \mathcal{C} \cup \{(\bar{x}^k, \bar{y}^k)\}$, $\mathcal{I}_d := \mathcal{I}_d \cup \{(\hat{x}^k, \hat{y}^k)\}$
      end
    end
    Continue with branch-and-cut tree search.
  end
end

Algorithm 8: Branch-and-Pump Algorithm

3.5 Numerical Results

The rounding-based feasibility pump has been built on top of FilMINT. The other two versions of feasibility pump have been implemented currently as standalone using the FilMINT framework. We also implemented, from scratch, the algorithm of Bonami et al. [2006] so as to be able to show the effect of our suggested improvements.

The tests have been performed using 64-bit, 1.8 GHz AMD Opteron microprocessors. The machines have 2 Gb of RAM and run Fedora Core 2 as the operating system. The codes that we wrote were compiled using the GNU suite of C, C++, and FORTRAN compilers. We ran the feasibility pump heuristics on 47 challenging convex MINLP instances. The test-set includes instances from the GAMS collection of MINLP problems (Bussieck et al. [2003]), the MacMINLP collection of test problems (Leyffer [2003]), and the collection by Sawaya et al. [2006].

We present tables of results showing the performance of the various algorithms. We also use performance profiles (see Dolan and Moré [2002]) to summarize and compare the runs. We use a scaled solution value as the metric for the comparison. We define the scaled solution value of instance i using algorithm s as $\rho_i^s = 1+(z_i^s - z_i^*)/z_i^*$, where z_i^s is the best solution value obtained by the implementation of algorithm s on instance i, and z_i^* is the best known solution value for instance i. The performance profile therefore indicates the quality of the solution found by the implementation of the algorithm in the time limit.

3.5.1 Results for the Rounding-based Feasibility Pump

We run the rounding-based feasibility pump (RFP) heuristic for a time limit of 90 seconds. We turn on SOS-based rounding whenever SOS-1 type variables are detected

by FilMINT. Our computational experience suggests that random quadratic rounding does better when compared to rounding to nearest integer and random threshold rounding. We therefore use random quadratic rounding in our experiments. We set T_0, the mean number of flipped variables, to 10. σ_{min} is set to be 0.02. The perturbation parameter P is set to be 0.4. We keep a history of $H = 4$ points to detect cycling. Also, α, the parameter that is used to improve the current incumbent, is set to be 0.05. These parameter settings are similar to those obtained by Fischetti et al. [2005] for their rounding-based feasibility pump scheme for MILPs.

We compare the solution found by rounding-based feasibility pump with FilMINT. Table 3.1 summarizes the results of the run. The table shows the solution value and time taken for the first solution and the best solution (in 90 seconds) for both solvers. A value of -1 in the table means that no feasible solution was found by the solver. For the 47 instances in the testsuite, both the solvers find at least one solution for 22 instances. RFP is the only solver to find a solution for four of the instances, while it fails in 17 instances. These 17 instances are synthesis (Sy*) and retro-synthesis (RS*) problems. When run for a time limit for 90 seconds, RFP gives the best solution in 12 instances, while FilMINT does better for 35 instances. However, when one compares the statistics for the first solution, RFP does well comparatively. It finds the better solution in 17 out of the 47 instances and takes lesser time for the first solution in 20 out of the 47 instances. Further, when one looks at the 22 instances where both solvers find a solution, RFP does even better. It finds the first solution in lesser time in 16 of the 22 instances. Also, the solution value obtained is better in 13 out of the 22 instances.

We now compare FilMINT with a version of FilMINT that is enhanced with the rounding-based feasibility pump. In this experiment, RFP is run in the beginning

of the branch-and-cut scheme of FilMINT for 90 seconds. We call this version of FilMINT as FilMINT++. We run these experiments for a time limit of 4 hours. Table 3.2 summarizes these results. We also show a performance profile (Figure 3.4) which uses time as the metric for comparing FilMINT with FilMINT++.

Table 3.1: Comparing Rounding-based Feasibility Pump with FilMINT (time limit of 90 seconds).

	Rounding based Feasibility Pump				FilMINT (default)			
Instance	first sol	time	best sol	time	first sol	time	best sol	time
BS1164	809817.2	0.1	769629.3	31.6	1007389.8	0.7	729948.7	76.3
BS1164	809817.2	0.1	769629.3	31.6	1007389.8	0.7	735198.7	8.7
BS210M	2360025.3	0.6	2360025.3	0.6	2441984.6	1.7	2303725.8	53.0
fo7	49.0	0.4	25.8	10.1	33.7	27.1	22.9	72.3
fo72	41.9	0.2	18.9	82.4	18.9	10.4	17.8	80.4
fo8	39.0	0.8	26.9	80.7	35.1	41.8	32.4	57.2
fo9	65.2	2.4	48.4	26.7	-1	-1	-1	-1
m7	392.0	0.3	147.4	9.7	220.5	0.0	148.6	58.1
o7	171.3	0.5	136.6	50.4	151.3	78.5	151.3	78.5
o72	164.7	0.1	137.2	37.3	128.9	29.1	120.7	85.0
tls6	22.1	1.0	22.1	1.0	-1	-1	-1	-1
tls7	-1	-1	-1	-1	-1	-1	-1	-1
tls12	-1	-1	-1	-1	-1	-1	-1	-1
trlss4	13.0	0.1	10.0	0.2	12.6	64.0	9.7	79.2
trlss5	13.0	0.2	11.5	0.4	-1	-1	-1	-1
trlss6	27.1	1.3	18.1	59.8	-1	-1	-1	-1

Continued on next page

Continued from previous page

	Rounding based Feasibility Pump				FilMINT (default)			
Instance	first sol	time	best sol	time	first sol	time	best sol	time
trlss7	-1	-1	-1	-1	-1	-1	-1	-1
trls12	-1	-1	-1	-1	-1	-1	-1	-1
safeCH	37200.4	23.3	37200.4	23.3	41538.6	31.3	29813.9	60.2
Safe3	242902.6	0.1	41610.7	53.6	242147.7	0.0	34560.0	40.3
FLay5M	64.5	0.0	64.5	0.0	64.5	0.0	64.5	0.0
FLay5H	64.5	0.1	64.5	0.1	64.5	0.1	64.5	0.1
FLay6H	66.9	0.8	66.9	0.8	66.9	0.1	66.9	0.1
FLay6M	66.9	0.0	66.9	0.0	66.9	0.0	66.9	0.0
RS83H	-1	-1	-1	-1	-597.9	0.5	-2159.5	74.7
RS832M	-1	-1	-1	-1	-38.1	0.1	-537.1	36.9
RS833M	-1	-1	-1	-1	-256.7	0.3	-833.2	82.5
RS834M	-1	-1	-1	-1	-597.9	0.5	-2159.5	74.6
RS83M	-1	-1	-1	-1	-210.2	0.0	-505.7	21.6
RS84H	-1	-1	-1	-1	-597.9	0.6	-1236.1	85.6
RS842M	-1	-1	-1	-1	-38.1	0.2	-566.4	67.1
RS843M	-1	-1	-1	-1	-1480.8	0.5	-2618.2	76.8
RS844M	-1	-1	-1	-1	-597.9	0.6	-1236.1	85.6
RS84M	-1	-1	-1	-1	-58.4	0.0	-281.6	23.7
SL08H	293990.3	1.0	123628.3	67.1	311171.0	0.2	89100.5	14.4
SL09M	268502.3	0.0	126427.3	35.6	328152.3	0.0	110890.5	52.3
SL10M	227047.3	0.1	181197.2	45.5	365467.3	0.1	150998.7	1.7
SL09H	448927.9	1.9	136300.3	76.5	529177.1	0.3	116898.8	31.7
SL10H	524182.9	4.6	524182.9	4.6	502886.6	0.4	180037.2	45.2

Continued on next page

Continued from previous page

	Rounding based Feasibility Pump				FilMINT (default)			
Instance	first sol	time	best sol	time	first sol	time	best sol	time
Sy2M3M	-1	-1	-1	-1	-22.3	0.0	-2647.0	76.7
Sy2M4M	-1	-1	-1	-1	-41.1	0.1	-3511.0	17.7
Sy3M2M	-1	-1	-1	-1	-16.6	0.0	-395.3	53.7
Sy3M3M	-1	-1	-1	-1	-22.3	0.1	-627.2	48.3
Sy3M4M	-1	-1	-1	-1	-41.1	0.1	-804.3	2.6
Sy4M2M	-1	-1	-1	-1	-16.5	0.0	-334.2	74.1
Sy4M3M	-1	-1	-1	-1	-22.3	0.1	-305.4	7.0
stock	318270.9	0.0	123246.0	59.7	436419.1	0.0	176865.2	55.0

Table 3.2: FilMINT vs FILMINT with feasibility pump (4 hours). $*$ means proven optimality, -1 means no solution found.

	FilMINT (default)		FilMINT++	
Instance	solution	time	solution	time
BS1BM164	729948.7*	573.0	729948.7*	730.5
BSBM1164	729948.7*	759.0	729948.7*	716.2
BS21210M	2295348.8*	435.1	2295348.8*	391.7
fo7	20.7*	1975.8	20.7*	1290.9
fo72	17.7*	399.8	17.7*	879.7
fo8	22.4*	4821.2	22.4*	2780.0
fo9	23.5*	11036.4	23.5	14400.0
m7	106.8*	275.5	106.8*	214.0

Continued on next page

Continued from previous page

	FilMINT (default)		FilMINT++	
Instance	solution	time	solution	time
o7	131.7	14400.0	134.2	14400.0
o72	116.9*	6437.6	116.9*	7090.2
tls6	-1	14400.1	22.1	14400.2
tls7	-1	14400.0	-1	14400.0
tls12	-1	14401.1	-1	14401.2
trimloss4	8.3*	1097.8	8.3*	1486.9
trimloss5	11.0	14400.1	11.0	14400.1
trimloss6	20.2	14400.0	18.1	14400.2
trimloss7	-1	14400.1	-1	14400.1
trimloss12	-1	14402.7	-1	14400.0
safetyCH	28380.7*	613.8	28380.7*	437.9
Safety3	27803.0	14400.0	27803.0*	13305.0
FLay05M	64.5*	1090.9	64.5*	1180.4
FLay05H	64.5*	2097.4	64.5*	2272.8
FLay06H	66.9	14400.1	66.9	14400.0
FLay06M	66.9	14400.0	66.9	14400.0
RS83H	-2350.0	14400.1	-2476.6	14400.1
RS83M2M	-702.0	14400.2	-716.5	14400.2
RS83M3M	-1516.0	14400.1	-1525.1	14400.0
RS83M4M	-2350.0	14400.1	-2476.6	14400.0
RS83M	-510.1*	1633.0	-510.1*	1616.0
RS84H	-2414.3	14400.1	-2404.4	14400.1
RS84M2M	-665.6	14400.1	-649.5	14400.1

Continued on next page

Continued from previous page				
	FilMINT (default)		FilMINT++	
Instance	solution	time	solution	time
RS84M3M	-2688.7	14400.0	-2688.7	14400.0
RS84M4M	-2414.3	14400.1	-2404.4	14400.1
RS84M	-311.6	14400.1	-266.2	14400.0
SLay08H	84960.2*	952.9	84960.2*	946.1
SLay09M	107805.8*	484.3	107805.8*	355.5
SLay10M	129579.9	14400.0	129579.9*	8356.5
SLay09H	108805.4	14400.0	107805.8*	5001.5
SLay10H	138403.9	14400.0	502886.6	14400.0
Sy2M3M	-2647.0*	416.9	-2647.0*	524.2
Sy2M4M	-3532.7*	2938.0	-3532.7*	3091.2
Sy3M2M	-399.7*	1278.7	-399.7*	1365.8
Sy3M3M	-652.7	14400.0	-652.7	14400.0
Sy3M4M	-856.2	14400.0	-848.2	14400.0
Sy4M2M	-387.3	14400.0	-387.3	14400.0
Sy4M3M	-333.4	14400.0	-333.4	14400.0
stock	130540.7	14400.0	123246.0	14400.1

FilMINT++ solves 21 of the 47 problems within the time limit, while FilMINT solves only 19 instances within the time limit. The profile shows that the use of RFP causes a reduction in the time taken to solve. Further, the use of rounding-based feasibility pump leads to a solution for the instance tls6, which was not known earlier.

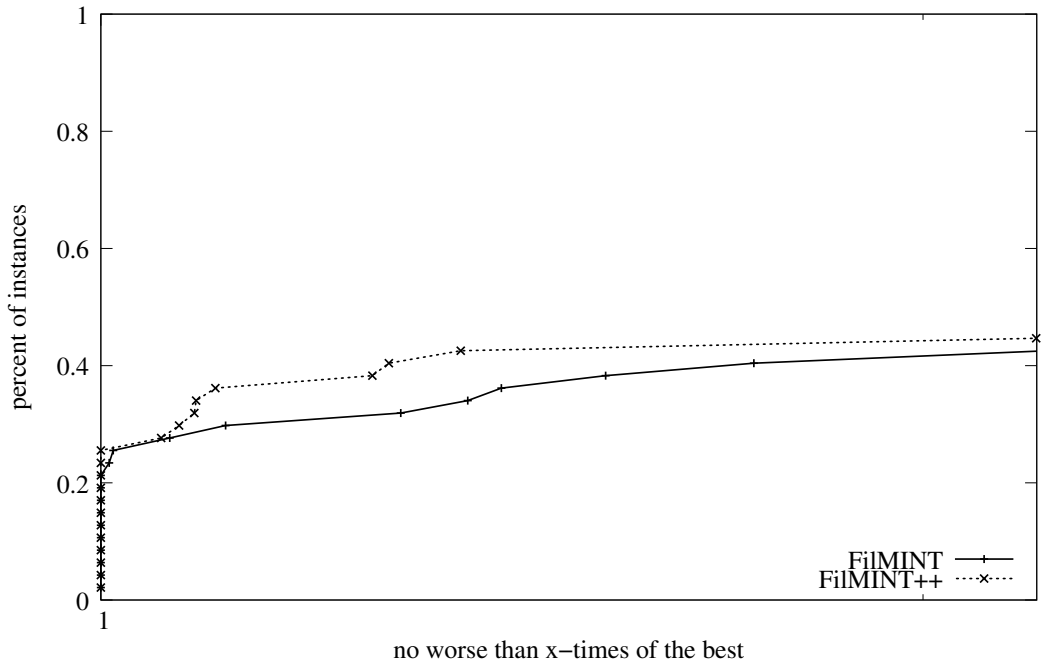

Figure 3.4: Performance profile comparing FilMINT with FilMINT enhanced with RFP

The results show that rounding-based feasibility pump may indeed be useful for getting an integer feasible solution for instances where it is hard to find any solution. The use of rounding-based pump as a heuristic in a branch-and-cut setting can lead to reduction in solution time. Further, rounding-based pump gives a solution quickly, and should be run for a small time limit when used as a heuristic in a branch-and-cut scheme. Compared to MILP-based feasibility pump, this scheme is cheap to solve, as rounding is inexpensive. We however note that the rounding-based pump may fail for some instances where it gets stuck due to cycling. One should also investigate different rounding schemes.

3.5.2 Results for the MILP-based Feasibility Pumps

We now investigate the MILP-based feasibility pumps proposed in section 3.3 and 3.4. We implemented the algorithm of Bonami et al. [2006] so as to be able to compare it

with our proposed schemes. The two proposed schemes and the scheme by Bonami et al. [2006] have been implemented currently as standalone applications. We run these three different versions of the MILP-based feasibility pump for a time limit of 90 seconds and compare solution value, time, and iterations required for the first incumbent and the best incumbent within this time. We set the improvement factor α to be 0.025 for each of the three algorithms.

Table 3.3 shows the results for the first solution found by the implementation of the MILP-based pump by Bonami et al. [2006], modified MILP-based feasibility pump, and branch-and-pump. We refer to implementation of algorithm of Bonami et al. [2006] as BONFP from now on. The implementation of our modified version of MILP-based feasibility pump is referred as MFP, and the implementation of branch-and-pump as BNP from now on. Table 3.4 shows the results for the best solution found by BONFP, MFP, and BNP within the time limit of 90 seconds. Again, as before, the value of -1 in these tables signifies that no integer feasible solution was found by the solver. In each of the tables, column 1 shows the instance name, columns 2-3 show the solution value and time by BONFP, columns 4-5 show the solution value and time by MFP, and columns 6-7 show the solution value and time by BNP.

Table 3.3: Table of results comparing first solution found by BONFP, MFP and BNP.

	BONFP		MFP		BNP	
Instance	first sol	time	first sol	time	first sol	time
BS1BM164	1414989.6	0.4	898809.6	0.3	898809.6	0.3
BSBM1164	1414989.6	0.4	898809.6	0.3	898809.6	0.3

Continued on next page

Continued from previous page

	BONFP		MFP		BNP	
Instance	first sol	time	first sol	time	first sol	time
BS21210M	4015248.7	1.5	2487129.2	1.5	2487129.2	1.5
fo7	39.4	0.4	39.4	0.4	38.9	3.9
fo72	33.3	5.7	39.4	3.5	47.2	10.9
fo8	65.3	20.0	45.9	16.3	73.6	47.5
fo9	66.7	8.2	65.3	36.1	-1	-1
m7	220.5	0.1	220.5	0.1	220.5	0.1
o7	167.3	1.6	165.4	0.9	174.8	12.5
o72	148.9	6.0	175.8	2.6	199.2	1.6
tls6	-1	-1	22.0	6.6	24.9	30.3
tls7	-1	-1	-1	-1	-1	-1
tls12	-1	-1	-1	-1	-1	-1
trimloss4	19.6	0.2	19.6	0.1	11.5	0.5
trimloss5	21.6	2.1	27.5	0.4	15.9	41.1
trimloss6	32.4	8.5	48.1	1.2	27.9	19.6
trimloss7	-1	-1	-1	-1	-1	-1
trimloss12	-1	-1	-1	-1	-1	-1
safetyCH	79179.3	6.9	58597.4	1.7	-1	-1
Safety3	284110.0	0.3	269718.4	0.1	269718.4	0.1
FLay05M	120.0	0.0	64.5	0.0	64.5	0.0
FLay05H	120.0	0.2	64.5	0.2	64.5	0.2
FLay06H	120.0	0.5	66.9	0.5	66.9	0.5
FLay06M	120.0	0.1	66.9	0.0	66.9	0.0
RS83H	566.6	4.6	-597.9	2.9	-597.9	2.9

Continued on next page

Continued from previous page						
	BONFP		MFP		BNP	
Instance	first sol	time	first sol	time	first sol	time
RS83M2M	328.4	1.1	-68.1	0.7	-68.1	0.7
RS83M3M	444.6	2.3	-286.7	1.6	-286.7	1.6
RS83M4M	566.6	4.6	-597.9	2.9	-597.9	2.9
RS83M	90.1	0.1	-260.8	0.1	-260.8	0.1
RS84H	953.3	5.3	-600.9	4.2	-600.9	4.1
RS84M2M	447.2	1.6	-68.5	0.9	-68.5	0.9
RS84M3M	119.1	3.3	-1666.9	2.3	-1666.9	2.3
RS84M4M	953.3	5.3	-600.9	4.2	-600.9	4.1
RS84M	173.4	0.2	-109.0	0.1	-109.0	0.1
SLay08H	364135.0	1.0	203143.5	0.5	203143.5	0.5
SLay09M	524232.5	0.2	344702.3	0.1	344702.3	0.1
SLay10M	545890.0	0.3	436218.4	0.2	436218.4	0.2
SLay09H	909091.3	2.1	491242.7	1.1	491242.7	1.1
SLay10H	1224023.8	4.2	676939.6	4.4	676939.6	4.3
Sy2M3M	5.0	0.2	-17.1	0.2	-17.1	0.2
Sy2M4M	5.0	0.3	-31.1	0.3	-31.1	0.3
Sy3M2M	5.0	0.2	-13.1	0.2	-13.1	0.2
Sy3M3M	5.0	0.4	-17.1	0.4	-17.1	0.4
Sy3M4M	5.0	0.7	-31.1	0.7	-31.1	0.7
Sy4M2M	5.0	0.3	-13.1	0.3	-13.1	0.3
Sy4M3M	5.0	0.7	-17.1	0.7	-17.1	0.7
stock	435633.2	0.1	435633.2	0.1	435633.2	0.1

Table 3.4: Table of results comparing best solution found by BONFP, MFP and BNP within 90 seconds.

	BONFP		MFP		BNP	
Instance	best sol	time	best sol	time	best sol	time
BS1BM164	843988.1	88.8	733934.4	85.9	741953.2	53.2
BSBM1164	851441.4	89.0	760863.4	88.4	700855.2	53.1
BS21210M	2469281.1	59.5	2298404.4	85.2	2299722.9	70.4
fo7	36.4	70.4	22.5	3.0	21.0	74.0
fo72	31.5	67.0	17.7	55.4	20.8	52.1
fo8	64.0	78.8	40.4	77.5	45.6	62.5
fo9	66.0	29.4	37.9	73.0	-1	-1
m7	185.1	85.9	106.8	3.9	106.8	8.6
o7	148.7	83.5	138.9	2.4	142.1	15.9
o72	146.0	89.9	129.1	16.0	124.2	59.2
tls6	-1	-1	17.0	12.3	17.9	89.9
tls7	-1	-1	-1	-1	-1	-1
tls12	-1	-1	-1	-1	-1	-1
trimloss4	9.5	22.6	8.4	54.7	8.4	71.1
trimloss5	15.0	35.9	10.7	26.1	12.0	73.8
trimloss6	29.5	49.7	20.1	43.4	18.4	45.5
trimloss7	-1	-1	-1	-1	-1	-1
trimloss12	-1	-1	-1	-1	-1	-1
safetyCH	50040.5	17.3	28380.7	11.9	-1	-1
Safety3	68007.5	89.6	28527.3	26.6	38894.6	86.7
FLay05M	64.6	4.7	64.5	0.0	64.5	0.0

Continued on next page

Continued from previous page

	BONFP		MFP		BNP	
Instance	best sol	time	best sol	time	best sol	time
FLay05H	64.6	8.7	64.5	0.2	64.5	0.2
FLay06H	67.0	19.9	66.9	0.5	66.9	0.5
FLay06M	67.2	51.7	66.9	0.0	66.9	0.0
RS83H	-822.2	89.7	-744.6	37.0	-744.6	40.6
RS83M2M	-262.0	86.1	-135.9	25.5	-644.9	41.9
RS83M3M	-502.8	44.9	-473.5	46.0	-753.4	76.9
RS83M4M	-822.2	89.4	-744.6	37.0	-744.6	40.6
RS83M	-315.4	8.8	-493.8	10.6	-492.1	5.1
RS84H	-868.7	89.3	-746.5	74.9	-600.9	4.1
RS84M2M	-152.7	51.4	-522.8	84.9	-515.9	25.0
RS84M3M	-1361.6	87.5	-2491.3	18.8	-2491.3	19.3
RS84M4M	-868.7	88.6	-746.5	75.4	-600.9	4.1
RS84M	-113.4	67.2	-309.3	1.7	-309.3	1.9
SLay08H	357096.9	89.0	84960.2	9.5	84960.2	11.0
SLay09M	256692.1	79.1	107805.8	4.9	107805.8	6.9
SLay10M	297423.7	84.6	129813.7	80.1	129582.0	66.7
SLay09H	439339.7	88.0	107805.8	58.2	108870.7	82.1
SLay10H	667957.9	85.9	134364.5	74.2	164096.3	88.8
Sy2M3M	-2174.6	21.5	-2642.0	5.9	-2601.5	2.0
Sy2M4M	-2887.1	32.0	-3471.0	22.9	-3473.9	8.0
Sy3M2M	-324.5	6.5	-398.3	29.8	-380.0	26.3
Sy3M3M	-584.3	8.6	-629.9	65.6	-313.1	24.1
Sy3M4M	-747.7	23.6	-800.5	77.6	-364.1	18.7

Continued on next page

Continued from previous page						
	BONFP		MFP		BNP	
Instance	best sol	time	best sol	time	best sol	time
Sy4M2M	-323.5	17.8	-347.8	27.7	-343.5	82.9
Sy4M3M	-233.1	24.3	-200.0	58.4	-176.5	26.0
stock	196946.7	40.0	130893.5	57.9	128677.5	80.2

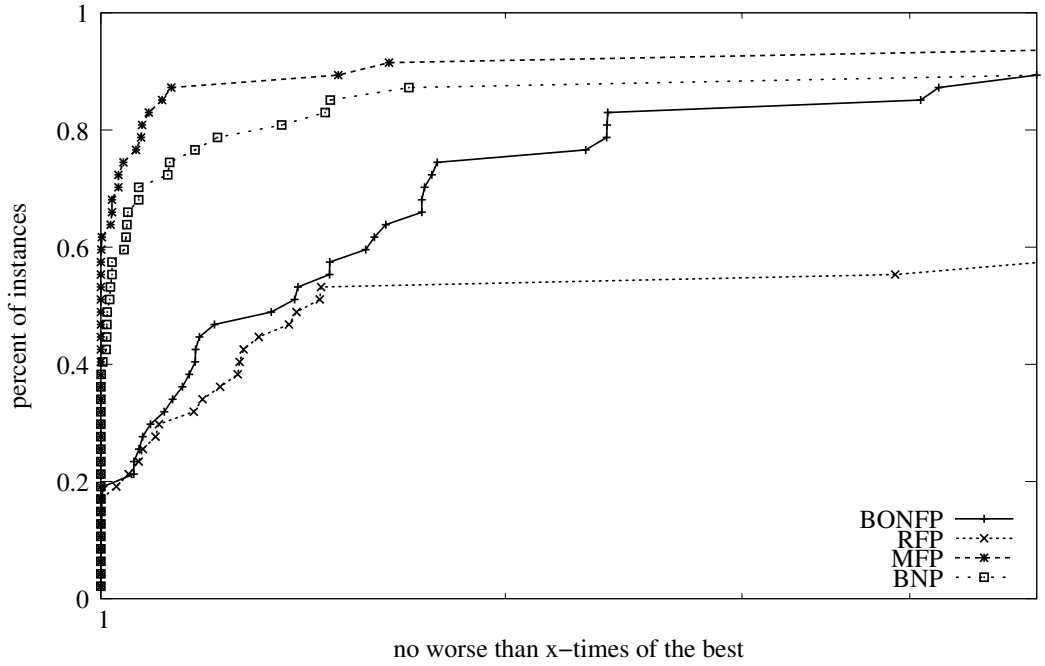

Figure 3.5: Performance profile comparing rounding-based pump with the MILP-based pumps run for 90 seconds (solution value as the metric).

We next summarize the results in Tables 3.3–3.4. BONFP finds integer feasible solutions for all but 5 instances, whereas MFP finds for all but 4 instances. BNP does not find solution for 6 instances. Comparison of the results for getting the first integer feasible solution shows that BNP and MFP do much better than BONFP. MFP gets the best first solution in 37 out of the 47 instances, whereas BNP gets the best first

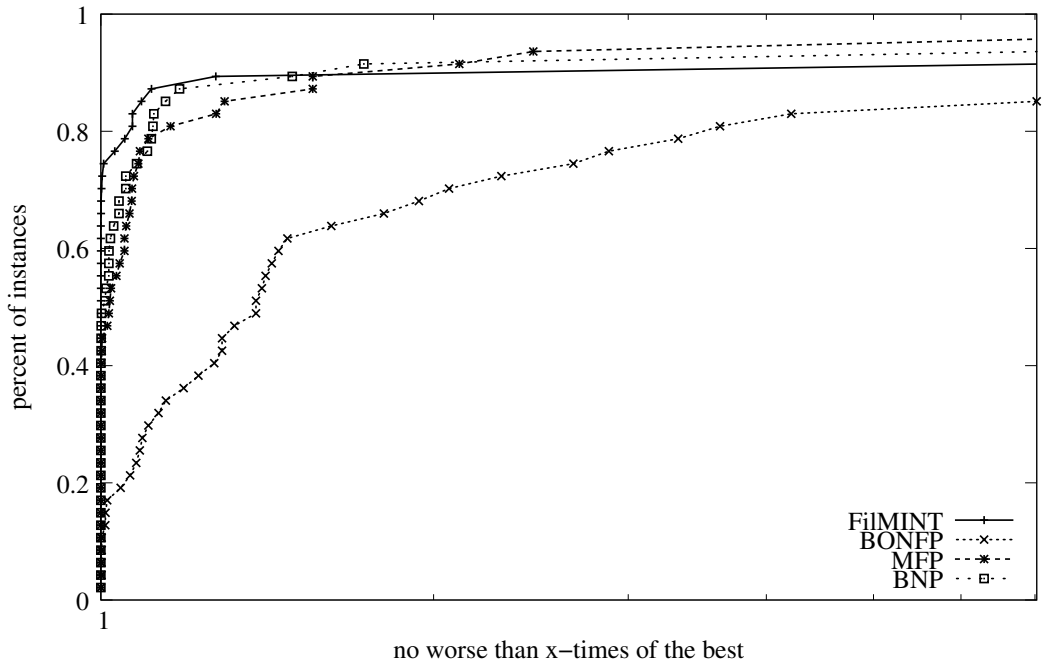

Figure 3.6: Performance profile comparing FilMINT with the MILP-based pumps run for 4 hours (solution value as the metric).

solution in 36 cases. In comparison, BONFP gets the best first solution in only 4 instances. For the time taken to get the first solution, MFP wins on 37 instances again, BNP on 36, and BONFP in 17 instances. The comparison of the results at the end of 90 seconds shows that MFP gets the best solution for 28 instances, followed by BNP in 20, and then BONFP in 5 instances. Compared to BONFP, BNP and MFP improve the solution value by a factor of 2-3 on an average. Similarly, BNP and MFP reduce the time taken to get the first and the best solution by a factor of 3-4. The difference in the results is even more pronounced for retro-synthesis (RS*) and synthesis (Sy*) instances.

The performance profile (Figure 3.5) compares the best solution value found by the three algorithms in a time limit of 90 seconds. The relative performance of rounding-based feasibility pump (RFP) is also seen from the profile. The profile shows that MFP does the best, closely followed by BNP. We feel that BNP does not do as well

or better than MFP because we do not change the objective of the MILP each time a distance-based NLP is solved. A close look at Figure 3.5 shows that RFP gets the best solution for 20% of the instances. This is quite promising since it does not have to solve costly MILPs during the iterative procedure.

Finally, we compare the performance of the two MILP-based algorithms when run for a time limit of 4 hours. We note that these algorithms are exact solution procedures for convex MINLP problems, and so can be used to solve convex MINLPs to optimality. Figure 3.6 shows the performance profile comparing the solution values for instances solved with FilMINT, BONFP, MFP, and BNP. The profile shows that MFP and BNP compare favorably with FilMINT. They also give integer feasible solutions for more instances. Table 3.5 summarizes the results of the 4 hour runs on the hardest set of instances. No integer feasible solutions were previously known for 5 out of the 6 instances, with the exception of the work by Bonami et al. [2006]. MFP now finds integer feasible solutions for all instances except tls12. Also, MFP and BNP take much lesser time to get similar solutions compared to BONFP and FilMINT.

Table 3.5: Table of results comparing FILMINT and FP by Bonami et al. [2006] with modified FP and branch-and-pump run with 4 hours time limit

	FilMINT		BONFP		MFP		BNP	
Instance	z^*	time	z^*	time	z^*	time	z^*	time
tls6	-1	-1	-1	-1	16.2	11389.0	15.9	1978.7
tls7	-1	-1	-1	-1	18.7	11769.4	17.4	7908.5
tls12	-1	-1	-1	-1	-1	-1	-1	-1
trimloss6	20.2	2268.7	16.9	13392.1	16.2	179.1	15.9	547.4
trimloss7	-1	-1	46.7	8576.7	16.2	2604.5	16.4	12808.4
trimloss12	-1	-1	-1	-1	155.8	14198.6	-1	-1

3.6 Conclusions

We investigated different algorithmic schemes for finding integer feasible solution for MINLPs. We extended the heuristic by Fischetti et al. [2005] for MILPs to MINLPs. We note that these ideas may be used to extend other heuristics like local branching, and RINS as well. We improved the MILP based feasibility pump by Bonami et al. [2006] by improving the MILP formulation and schemes for improving integer feasible solutions iteratively. We also proposed the branch-and-pump algorithm, that follows an LP/NLP-like approach for getting integer feasible solutions. We note that the algorithm may be further improved by finding a way to change the objective whenever a distance-based NLP is solved during an MILP run. This may be a good direction for future research. Our computational results show the effectiveness of our proposed algorithms. We improve the solution quality and time taken for getting integer feasible solutions by a factor of 2-4. We now get integer feasible solutions for all except 1 instance in our test suite.

Chapter 4

Modeling without Categorical Variables: A Mixed-Integer Nonlinear Program for the Optimization of Thermal Insulation Systems

Optimal design applications are often modeled by using categorical variables to express discrete design decisions, such as material types. A disadvantage of using categorical variables is the lack of continuous relaxations, which precludes the use of modern integer programming techniques.

In this chapter, we show how to model such mathematical programs containing categorical variables as MINLPs, using standard integer modeling techniques. We also highlight some other common difficulties involved in modeling such systems,

including discontinuities, and functions for which gradients are difficult to compute. We illustrate this approach on a load-bearing thermal insulation system. The system consists of a number of insulators of different materials and intercepts that minimize the heat flow from a hot surface to a cold surface. We come up with three different MINLP models, with varying degrees of smoothness. Our new model allows us to employ modeling languages and solvers based on sophisticated solution techniques, and illustrates the interplay between integer and nonlinear modeling techniques. We present numerical experience that illustrates the advantage of the standard MINLP model.

4.1 Introduction

Recently, researchers have expressed interest in mixed-variable optimization problems (MVPs). Problems of this class involve *categorical variables*, which are constrained to take values from a finite set of values, including non-numerical values. MVPs have been used to design load-bearing thermal insulation systems, where the categorical variables model the type of material chosen for the insulators (Kokkolaras et al. [2001], Abramson [2004]). Categorical variables are a convenient way to move from a simulation tool that requires input such as material properties to an optimization tool. On the other hand, the presence of categorical variables precludes the use of modern integer optimization techniques, because continuous relaxations of the categorical decisions are not readily available.

We give an example of a model with categorical variables and show how the categorical variables can be replaced by standard integer modeling techniques. We illustrate our approach on an example of thermal insulation systems and emphasize

the interaction of integer and nonlinear modeling techniques. Our approach provides a blueprint for reformulating other design problems that involve categorical variables, for example the design of nanomaterials (Zhao et al. [2005]), and in optimal sensor placement (Beal et al. [2006]), thereby increasing the domain of applications that can be modeled and solved by applying MINLP solution techniques. We believe that the conclusions of this work also are relevant to application scientists who develop simulation tools. In our view, it is important to include optimization considerations in simulation tools from the start.

This chapter is organized as follows. In Section 4.2, we start by reviewing the categorical variable formulation of a thermal insulation system. In Section 4.3, we introduce the integer and nonlinear modeling techniques that helps us to reformulate this model as a standard MINLP. In Section 4.4, we obtain two MINLP models with varying degree of smoothness and comment on the relative merits of these formulations. In Section 4.5, we come up with a smooth MINLP model that allows us to remove some of the nonsmoothness from the previous models. In Section 4.6, we present numerical results illustrating the benefit of our new approach.

4.2 Load-Bearing Thermal Insulation Design

We consider the design of a load-bearing thermal insulation system. This system uses a series of heat intercepts and insulators to minimize the heat flow from a hot surface to a cold surface. The objective is to minimize the power required to maintain the heat intercepts at certain temperatures so that the cold surface can be maintained at the required temperature; see Figure 4.1. The insulator types are chosen from a set $\mathcal{M}$ of materials and are modeled as categorical variables.

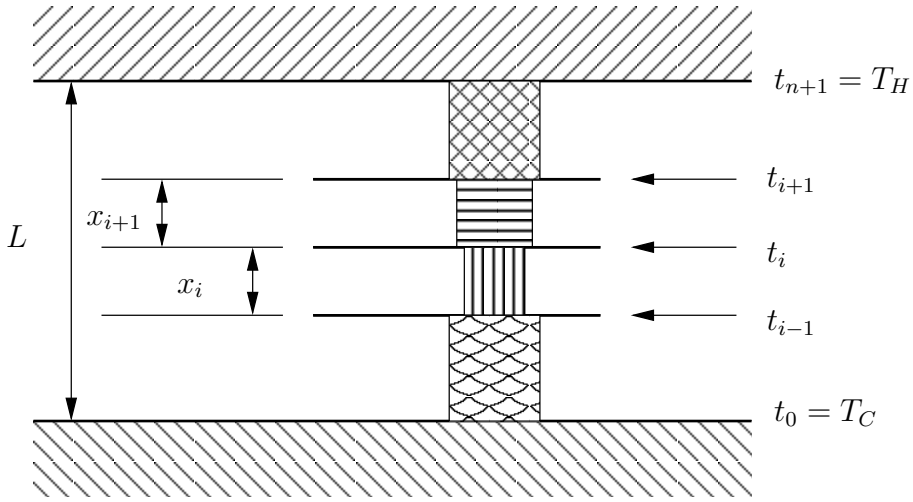

Figure 4.1: Illustration of the thermal insulation system.

The model is described in detail by Abramson [2004], who extends the model given by Kokkolaras et al. [2001] by adding load-bearing requirements. Thus, the insulators act as mechanical supports and must satisfy certain load-bearing constraints involving quantities such as thermal expansion, system mass, and stress.

4.2.1 Model Parameters and Data

The parameters and data of the model are summarized in Table 4.1. The thermodynamic cycle efficiency of intercept i is a piecewise constant function of the temperature (given in degrees Kelvin):

$$C(t_i) = \begin{cases} 5 & \text{if } t_i \leq 4.2, \\ 4 & \text{if } 4.2 < t_i < 71\,, \\ 2.5 & \text{if } t_i \geq 71. \end{cases} \quad i = 1, \ldots, n \tag{4.2.1}$$

The types of insulators are nylon, teflon, epoxy(normal), epoxy(plane), 6063-T5 aluminum, 1020 low-carbon steel, and 304 stainless steel. Their corresponding

Table 4.1: Model Parameters and Data

Parameter	Description	Value in Case Study
$C(t_i)$	thermodynamic cycle efficiency of intercept i	see (4.2.1)
$e(t, m)$	thermal expansion of insulator $m \in \mathcal{M}$ at temperature t	
$k(t, m)$	thermal conductivity of insulator $m \in \mathcal{M}$ at temperature t	
F	system load	250 kN
L	system length	10 cm
M	maximum system mass	10 kg
$\mathcal{M}$	set of insulator materials	see Table 4.2
N	maximum number of intercepts	10
T_C	cold surface temperature	4.2K
T_H	hot surface temperature	300K
δ	maximum thermal expansion	5%
$\rho(m)$	density of insulator $m \in \mathcal{M}$	see Table 4.2
$\sigma(t, m)$	tensile yield strength of insulator $m \in \mathcal{M}$ at temperature t	

densities, $\rho(m)$, are given in Table 4.2. Data for the thermal conductivity, $k(t, m)$, tensile yield strength, $\sigma(t, m)$, and thermal expansion, $e(t, m)$ are given in the form of look-up tables for every material $m \in \mathcal{M}$ and a discrete set of temperature values τ_j. Abramson [2004] and Kokkolaras et al. [2001] have fitted cubic splines to the data to provide a smooth approximation of these functions for every type of material. Our tables are also made available electronically as AMPL [Fourer et al., 1993] data files.

Table 4.2: Densities for the Various Insulator Materials

Nylon	Teflon	Epoxy-normal	Epoxy-plane	Aluminum	Steel	Carbon-steel
0.0010	0.0015	0.0018	0.0018	0.0027	0.0078	0.0078

4.2.2 Model Variables

We summarize the definition of the model variables in Table 4.3. Throughout, we index the intercepts by subscripts $i = 0, \ldots, n + 1$, where index $i = 0$ corresponds to the cold surface and index $i = n + 1$ corresponds to the hot surface. The material

types are indexed by subscripts $j = 1, \ldots, |\mathcal{M}|$.

Table 4.3: Model Variables

Variable	Description
a_i	area of insulator $i = 1, \ldots, n+1$
m_i	material $m_i \in \mathcal{M}$ of insulator $i = 1, \ldots, n+1$
n	number of intercepts, $n \in \{1, 2, \ldots, N\}$
q_i	heat flow from intercept i to $i-1$, for $i = 1, \ldots, n+1$
t_i	temperature at intercept $i = 0, \ldots, n+1$
Δx_i	thermal expansion of layer $i = 1, \ldots, n+1$

Power is applied at each intercept i at its cooling temperature $t_i, i = 1, \ldots, n$. The gap between the intercepts $i-1$ and i is filled in with insulator of thickness x_i. The temperature of the hot surface is given by $t_{n+1} = T_H$ and that of the cold surface by $t_0 = T_C$. The insulators used in the system may have different cross-sectional areas a_i. The design of the system involves choosing the number of intercepts n, their cooling temperatures t_i, the insulator types m_i, their thickness x_i and the cross-sectional areas a_i. We include the thermal expansion, $\Delta x_i / x_i$, for convenience but note that it is later eliminated from the model. The presence of categorical variables such as insulator types m_i and the number of insulators n make the model into a mixed-variable program. We note that even though n is an integer variable, it cannot take on fractional values because it appears as the upper summation bound in (4.2.4), (4.2.7), and (4.2.8). The fact that n is also an indexing bound in (4.2.3) and (4.2.5) implies that the number of model variables changes as n changes. Hence, we regard n as a categorical variable of the model.

4.2.3 Mixed-Variable Optimization Model

We can now state the complete mixed-variable optimization model.

$$\text{minimize} \quad \sum_{i=1}^{n} C(t_i)\left(\frac{T_H}{t_i} - 1\right) \cdot (q_{i+1} - q_i) \tag{4.2.2}$$

$$\text{subject to} \quad q_i = \frac{a_i}{x_i}\int_{t_{i-1}}^{t_i} k(t, m_i)dt, \quad i = 1, \ldots, n+1 \tag{4.2.3}$$

$$\sum_{i=1}^{n} \rho(m_i) a_i x_i \leq M \tag{4.2.4}$$

$$\frac{F}{a_i} \leq \bar{\sigma}_i = \min\{\sigma(t, m_i) \,:\, t_{i-1} \leq t \leq t_i\}, \quad i = 1, \ldots, n+1 \tag{4.2.5}$$

$$\sum_{i=1}^{n} \left(\frac{\Delta x_i}{x_i}\right)\left(\frac{x_i}{L}\right) \leq \frac{\delta}{100} \tag{4.2.6}$$

$$\frac{\Delta x_i}{x_i} = \frac{\int_{t_{i-1}}^{t_i} e(t, m_i)k(t, m_i)dt}{\int_{t_{i-1}}^{t_i} k(t, m_i)dt}, \quad i = 1, \ldots, n \tag{4.2.7}$$

$$\sum_{i=1}^{n} x_i = L \tag{4.2.8}$$

$$t_{i-1} \leq t_i \leq t_{i+1}, \quad i = 1, \ldots, n \tag{4.2.9}$$

$$t_0 = T_C \quad \text{and} \quad t_{n+1} = T_H \tag{4.2.10}$$

$$1 \leq n \leq N, \quad x_i \geq 0, \quad a_i \geq 0, \quad i = 1, \ldots, n+1 \tag{4.2.11}$$

The model contains five classes of nonlinear constraints. Equation (4.2.3) defines the heat flow from intercept i to $i-1$, which is governed by Fourier's law. Equation (4.2.4) is the mass constraint of the system. The stress of the system must not exceed the specified load F, which is modeled by Equation (4.2.5). The thermal expansion constraint is modeled by Equation (4.2.6), with the thermal contraction $\frac{\Delta x_i}{x_i}$ defined by the constraint (4.2.7). In addition, the model contains some linear

constraints: (4.2.8) constrains the thickness of the design, (4.2.9) orders the temperatures, and (4.2.10) fixes the temperatures at the cold and hot surface. We note that the latter two constraints imply that $T_C \leq t_i \leq T_H$ for $i = 0, \ldots, N+1$.

Abramson [2004] uses a different objective function in his `matlab` implementation, namely,

$$f_2 := \sum_{i=1}^{n+1} q_i \left(C(t_{i-1})(\frac{T_H}{t_{i-1}} - 1) - C(t_i)(\frac{T_H}{t_i} - 1) \right). \tag{4.2.12}$$

We use this objective function from now on for the purpose of modeling and for comparing our results with the work of Abramson.

The integrals in (4.2.3) and (4.2.7) are approximated by Simpson's rule, where the material specific functions $e(t, m_i)$ and $k(t, m_i)$ are derived from cubic spline interpolation of tabulated data.

Solving the thermal insulation problem involves optimizing a nonlinear objective function and nonlinear constraints over a variable space that includes categorical variables. The model is generally solved by using a pattern-search algorithm [Audet and Dennis, 2004, 2000, Abramson et al., 2004]. The computational burden of pattern-search techniques grows significantly with the number of variables, and this fact motivates the removal of as many defined variables as possible. For example, the fixed variables t_0 and t_{n+1} are removed. We can also remove the thermal expansion variables $\frac{\Delta x_i}{x_i}$ by substituting (4.2.7) into (4.2.6). In addition, it is argued in [Abramson, 2004] that the stress constraint must be binding at a solution, thereby implying that $a_i = \frac{F}{\bar{\sigma}_i}$ and allowing us to remove the variables a_i.

4.2.4 Challenges of the MVP Model

The MVP model (4.2.2)–(4.2.10) introduces a number of difficulties that appear to make it impossible to employ standard MINLP techniques to solve the problem:

1 The model contains a number of categorical variables that do not allow continuous relaxations. For example, it is not clear how to relax the condition that the materials be chosen from $\mathcal{M}$. Worse, the variable n appears as the upper limit in the summation, which makes constraints such as (4.2.8), (4.2.4), (4.2.6), and the objective function ((4.2.2) or (4.2.12)) discontinuous. Moreover, by changing n, we also change the number of variables t_i and so forth that appear in the model. In this sense, every n defines a different model.

2 The integrals in (4.2.3) and (4.2.7) cannot be evaluated exactly, because the integrands are only available as tabulated data. Thus, there exists no analytic closed-form expression for (4.2.3) and (4.2.7). Instead, the integrals are evaluated by using Simpson's rule. Thus, derivatives are difficult to compute, and hence derivative-free optimization techniques are used.

3 Constraint (4.2.5) contains a minimization for which no closed-form expression exists. The presence of this constraint results in a bilevel optimization problem that is considerably harder to solve. We note that this constraint can be written equivalently as an infinite set constraint, by requiring that

$$\frac{F}{a_i} \leq \sigma_i(t, m_i), \quad \forall t \in [t_{i-1}, t_i],\ \forall i = 1, \ldots, n+1.$$

4 The objective function is discontinuous because of the presence of the thermodynamic cycle efficiency coefficients, $C(t_i)$; see (4.2.1). This discontinuity can

cause derivative-based NLP solvers to fail.

Each of these points is a potential death-blow for standard optimization techniques. In [Abramson, 2004], a pattern-search technique is used to solve this problem. Pattern-search methods are a class of derivative-free optimization methods. They do not require gradients, or Hessians. Pattern-search techniques are easy to use, and often make progress with a limited budget of function evaluations.

Unfortunately, the presence of discontinuities implies that it is almost impossible to verify the optimality of a solution returned by the pattern-search method. Another drawback of pattern-search techniques is the fact that the quality of the solution depends on the definition of the neighborhood for the search. It is not hard to construct examples where the MVP pattern-search [Audet and Dennis, 2000] stalls at a pattern-optimal solution of integer points that is not a global minimum. Consider the following convex example:

$$\text{minimize } q(x) := 64(cx_1 - sx_2)^2 + (sx_1 - cx_2)^2 \quad \text{subject to } x_1, x_2 \text{ integer},$$

where $c = \cos(\pi/8),\ s = \sin(\pi/8)$ are rotational parameters. Figure 4.2 illustrates this situation: every pattern-center is a pattern-optimal solutions, yet clearly only one of them corresponds to the local/global minimum of $q(x)$. The example shows that pattern-search methods for mixed-variable problems may stall at points that are arbitrarily far from the global minimum. Fortunately, this does not seem to be the case with the thermal insulation problem, where pattern-search methods are competitive.

Next, we show how each of these challenges can be tackled by combining integer and nonlinear modeling techniques. These reformulations result in a standard MINLP

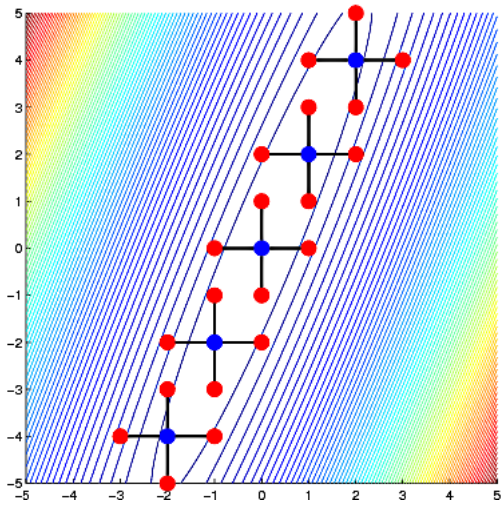

Figure 4.2: Example illustrating failure of MVP pattern-search.

that we formulate in the modeling language AMPL [Fourer et al., 1993]. The new formulation allows us to derive cutting planes and employ more powerful optimization techniques. In addition, we believe that by using a modeling language, our model becomes more transparent, ultimately enabling the design of larger and more complex systems.

4.3 Modeling Categorical Variables with Binary Variables

We start by showing that the categorical variables can be replaced by integer variables, and we develop a derivative-free model of thermal insulation that allows relaxations to be computed. We use binary indicator variables y_i to remove the variable n (the number of intercepts) from the summation bound in (4.2.4) and (4.2.6) and to eliminate the dependence of the number of decision variables on the decision variable n; we use binary decision variables y_i to denote the existence of layer i. This is a natural

reformulation and not necessarily inefficient in practice, since the maximum number of intercepts N is typically small. For the specific instance we solve in Section 4.6, $N = 20$. This reformulation is made by introducing the following inequality system:

$$\begin{aligned} \sum_{i=1}^{N+1} y_i &= n+1 && (4.3.13)\\ y_{i+1} &\leq y_i, \qquad i = 1, \ldots, N && (4.3.14)\\ x_i &\leq L y_i \qquad i = 1, \ldots, N+1 && (4.3.15)\\ y_i &\in \{0,1\}, \qquad i = 1, \ldots, N+1. \end{aligned}$$

The inequalities (4.3.14) order the intercepts and ensure that only consecutive intercepts in this ordering can exist. The variable upper bound inequalities (4.3.15) ensure that there is a positive thickness to the layer only if the layer is chosen to exist. The inequalities (4.3.15) can be replaced by the much stronger set of inequalities

$$\sum_{j=i}^{N+1} x_j \leq L y_i \qquad i = 1, \ldots, N+1. \tag{4.3.16}$$

In fact, Theorem 4.3.1 establishes that the inequalities (4.3.16) are facets of the convex hull of the inequality system of the reformulation.

Theorem 4.3.1. *Let*

$$P = \operatorname{conv}\left(\{(x,y) \in \mathbb{R}_+^{N+1} \times \mathbb{B}_+^{N+1} \mid y_{i+1} \leq y_i \quad i = 1, \ldots, N, \right.$$
$$\left. x_i \leq L y_i \quad i = 1, \ldots, N+1\}\right).$$

Then

$$\sum_{j=i}^{N+1} x_j \leq L y_i$$

defines a facet of P for each $i = 1, 2, \ldots, N+1$.

Proof. Assume that for each $i = 1, \ldots, N+1$ there is an inequality $\pi^i x + \mu^i y \leq \pi_0^i$ that is valid for P and satisfies

$$F_i \stackrel{\text{def}}{=} \{(x,y) \in P \mid \sum_{j=i}^{N+1} x_j = Ly_i\} \subseteq \{(x,y) \in P \mid \pi^i x + \mu^i y = \pi_0^i\} \stackrel{\text{def}}{=} \hat{F}_i.$$

In this case, we will show that $\pi^i x + \mu^i y \leq \pi_0^i$ is a scalar multiple of the inequality $\sum_{j=i}^{N+1} x_j \leq Ly_i$ which implies that $\sum_{j=i}^{N+1} x_j \leq Ly_i$ is a facet-defining inequality for P. (See [Nemhauser and Wolsey, 1988], Theorem I.4.3.5 for a proof of this result.)

First note that for each $i = 1, \ldots, N+1$, the point $(0,0) \in F_i$, so also the origin $(0,0) \in \hat{F}_i$, which implies that $\pi^i(0) + \mu^i(0) = \pi_0^i$, or

$$\pi_0^i = 0 \quad \forall i = 1, \ldots, N+1. \tag{4.3.17}$$

For any $i \geq 2$, the point $(0, e_1) \in F_i$. Since this point is also in $\hat{F}_i$, we have that $\mu_1^i = \pi_0^i = 0 \; \forall i \geq 2$. For any $i \geq 3$, the point $(0, e_1 + e_2) \in F_i$, so $\mu_1^i + \mu_2^i = \pi_0^i \; \forall i \geq 3$. Similarly, one can establish that

$$\mu_k^i = 0 \quad \forall i = 2, \ldots, N+1, \; \forall k \leq i-1. \tag{4.3.18}$$

For every $i = 1, \ldots, N+1$ the point $(Le_i, \sum_{j=1}^{i} e_j) \in F_i$. This implies that for each i, $L\pi_i^i + \sum_{j=1}^{i} \mu_j^i = \pi_0^i$, but by (4.3.18) and (4.3.17), this implies that

$$\mu_i^i = -L\pi_i^i \quad \forall i = 1, \ldots, N+1. \tag{4.3.19}$$

For any $i = 2, \ldots, N+1$ and for any $k = 1, \ldots, i-1$, the point

$$\left(Le_i + \sum_{j=1}^{k} Le_j, \sum_{j=1}^{i} e_j \right) \in F_i,$$

which implies that

$$L \sum_{j=1}^{k} \pi_j^i + L\pi_i^i + \sum_{j=1}^{i} \mu_j^i = 0.$$

This, coupled with (4.3.18) and (4.3.17), implies that

$$\sum_{j=1}^{k} \pi_j^i = 0 \quad \forall i = 2, \ldots, N+1, \ k = 1, \ldots, i-1. \tag{4.3.20}$$

Using (4.3.20) sequentially for $k = 1, 2, \ldots, i-1$, one can see that

$$\pi_k^i = 0 \quad \forall i = 2, \ldots, N+1, \ k = 1, \ldots, i-1. \tag{4.3.21}$$

For any $i = 1, \ldots, N$ and for any $k = i+1, \ldots, N+1$, the following three points lie on the face F_i:

- $(Le_i, \sum_{j=1}^{i} e_j)$,
- $(Le_k, \sum_{j=1}^{k} e_j)$, and
- $(L/2e_i, L/2e_k, \sum_{j=1}^{k} e_j)$.

Since these points also then must lie on $\hat{F}_i$ and the relations (4.3.18) and (4.3.17) hold, this fact implies that the equations

$$\begin{aligned} L\pi_i^i + \mu_i^i &= 0 \\ L\pi_k^i + \sum_{j=1}^{k} \mu_j^i &= 0 \end{aligned}$$

$$L/2\pi_i^i + L/2\pi_k^i + \sum_{j=1}^{k} \mu_j^i \quad = \quad 0$$

hold, which in turn implies that

$$\pi_k^i = \pi_i^i \quad \forall i = 1, \ldots, N, \ k = i+1, \ldots, N+1 \tag{4.3.22}$$

$$\sum_{j=i+1}^{k} \mu_j^i = 0 \quad \forall i = 1, \ldots, N, \ k = i+1, \ldots, N+1. \tag{4.3.23}$$

Using the relation (4.3.23) with $k = i+1$, one can establish that $\mu_{i+1}^i = 0$. Then, using (4.3.23) subsequently with $k = i+2, \ldots N+1$, one can establish that

$$\mu_k^i = 0 \quad \forall i = 1, \ldots, N, \ k = i+1, \ldots, N+1. \tag{4.3.24}$$

Collecting the relations (4.3.17), (4.3.18), (4.3.19), (4.3.21), (4.3.22), and (4.3.18), we see that indeed the inequality $\pi^i x + \mu^i y \leq \pi_0^i$ is a scalar multiple of the inequality $\sum_{j=i}^{N+1} x_j \leq L y_i$ for every $i = 1, 2, \ldots, N+1$, which completes the proof. □

Having indicator variables y_i representing the existence of layer i also makes it convenient to model material properties. We let $z_{ij} = 1$ if and only if the j^{th} material is chosen for layer i (where the ordering of the material is arbitrary). Otherwise, we set $z_{ij} = 0$. We can use the following constraints to model the fact that only existing layers choose a material type.

$$\sum_{j=1}^{|\mathcal{M}|} z_{ij} = y_i, \quad i = 1, \ldots, N+1. \tag{4.3.25}$$

The constraints (4.2.8), (4.2.4), and (4.2.6) that involve n as a summation limit

can now be written equivalently by using the new summation bound N and the indicator variables y_i.

Next, we model the constraints involving data functions such as thermal conductivity $k(t, m)$ by observing that

$$k(t, m_i) = \sum_{j=1}^{|\mathcal{M}|} z_{ij} k(t, m_j),$$

where (with some abuse of notation) m_i is the material used in layer i, while m_j is the j^{th} material in $\mathcal{M}$. We can now remove the categorical variables from the mixed-variable model and define the following mixed-integer model:

$$\text{minimize} \quad \sum_{i=1}^{N+1} q_i y_i \left(C(t_{i-1})(\frac{T_H}{t_{i-1}} - 1) - C(t_i)(\frac{T_H}{t_i} - 1) \right) \; := \; CoolingPower \tag{4.3.26}$$

$$\text{subject to} \quad q_i = \frac{a_i}{x_i} \int_{t_{i-1}}^{t_i} \sum_{1}^{|\mathcal{M}|} z_{ij} k(t, m_j) dt, \quad i = 1, \dots N+1 \tag{4.3.27}$$

$$\sum_{i=1}^{N+1} \sum_{j=1}^{|\mathcal{M}|} \rho_j z_{ij} a_i x_i \leq M, \tag{4.3.28}$$

$$\frac{F}{a_i} \leq \bar{\sigma}_i = \min_t \left\{ \sum_{j=1}^{|\mathcal{M}|} z_{ij} \sigma(t, m_j) \; : \; t_{i-1} \leq t \leq t_i \right\}, \quad i = 1, \dots, n+1 \tag{4.3.29}$$

$$\sum_{i=1}^{N+1} \left(\frac{\Delta x_i}{x_i} \right) x_i \leq L \frac{\delta}{100}, \tag{4.3.30}$$

$$\frac{\Delta x_i}{x_i} = \frac{\int_{t_{i-1}}^{t_i} \sum_{j=1}^{|\mathcal{M}|} z_{ij} e(t, m_j) k(t, m_j) dt}{\int_{t_{i-1}}^{t_i} \sum_{j=1}^{|\mathcal{M}|} z_{ij} k(t, m_j) dt}, \quad i = 1, \dots, N \tag{4.3.31}$$

$$\sum_{i=1}^{N+1} x_i = L, \tag{4.3.32}$$

$$x_i \geq \epsilon L y_i, \quad i = 1, \dots, N+1, \tag{4.3.33}$$

$$\sum_{j=i}^{N+1} x_i \leq L y_i, \quad i = 1, \ldots, N+1, \tag{4.3.34}$$

$$t_i - t_{i-1} \geq \epsilon y_i, \quad i = 1 \ldots N+1 \tag{4.3.35}$$

$$t_0 = T_C, \quad t_{N+1} = T_H \tag{4.3.36}$$

$$t_i \geq T_H \cdot (1 - y_{i+1}), \quad i = 1, \ldots, N \tag{4.3.37}$$

$$A_{\min} y_i \leq a_i \leq A_{\max} y_i, \quad i = 1 \ldots N+1, \tag{4.3.38}$$

where $\epsilon > 0$ is a small constant and ρ_j is the density of the j^{th} material.

In addition to the constraints (4.3.34), which ensure that the width of layer i, $x_i = 0$, is zero whenever that layer does not exist. We also add a variable lower bound (4.3.33) to eliminate spurious solutions with zero-thickness layers. We choose $\epsilon = 10^{-2}$, which can be interpreted as a manufacturing tolerance. Equation (4.3.37) fixes the temperatures t_i of the layers that do not exist. This constraint ensures that for the nonexisting layers $j \geq n+1$, the term in the objective function is canceled, because $T_H/t_j - 1 = 0$, for all $j \geq n+1$ ($t_j = T_H$). In addition, we model the condition that the temperatures at the intercepts are nondecreasing ($t_i \geq t_{i-1}$) by insisting that they are separated by at least a small positive amount (again this helps avoid spurious solutions) by Equation (4.3.35). Finally, the absence of a layer is modeled with (4.3.38), which sets the area a_i to zero, for any nonexisting layer, $i \geq n+2$.

Our model is a mixed-integer simulation model, owing to the presence of the inner minimization and the integrals. Below, we show that we can further refine this model by developing a fully explicit MINLP model.

4.4 A MINLP Model for Thermal Insulation Design

In this section we show that the simulation model of the previous section can be formulated as a smooth MINLP that can be expressed by using standard modeling tools such as AMPL.

4.4.1 Avoiding Bilevel Optimization Problems

We start by showing that the structure of the data allows us to avoid the bilevel constraints (4.2.5). It is argued in [Abramson, 2004] that the relationship between a_i and the power applied at an intercept i (given as $C(t_i)\left(\frac{T_H}{t_i}-1\right)\cdot(q_{i+1}-q_i)$) imply that the constraints (4.2.5) are always binding and, thus, the area variables are removed from the model. We are not sure that this argument holds in general; instead, we proceed in a different way.

We introduce a finer piecewise linear approximation of the data $\sigma(t,m)$ by adding data points from the cubic spline interpolation used by Abramson [2004], denoted by

$$T_C = T_1 < T_2 < \ldots < T_D = T_H, \tag{4.4.1}$$

where D is the number of discretization points (typically 20). Next, we approximate the minimization in (4.2.5) by requiring that the bound constraint holds at temperatures at each intercept. Thus we introduce the constraints

$$Fz_{ij} \le a_i\left(\sigma(T_r,m_j)+\frac{\sigma(T_{r+1},m_j)-\sigma(T_r,m_j)}{T_{r+1}-T_r}(t_i-T_r)\right), \tag{4.4.2}$$

for all materials $j = 1, \ldots, |\mathcal{M}|,\ i = 1, \ldots, N+1$, where the index r is such that $T_r \leq t_i < T_{r+1}$. We note that for most materials, $\sigma(t, m)$ is monotonic in t, and we need to enforce this bound only at the upper end of each intercept. However, the epoxy materials do not have monotonic $\sigma(t, m)$, and we must therefore enforce the bounds at both the lower and the upper intercept. We do this by adding the constraints at t_{i-1}

$$Fz_{ij} \leq a_i \left(\sigma(T_r, m_j) + \frac{\sigma(T_{r+1}, m_j) - \sigma(T_r, m_j)}{T_{r+1} - T_r}(t_{i-1} - T_r) \right), \tag{4.4.3}$$

for all materials $m_j \in \{\text{epoxy-n}, \text{epoxy-p}\}$ and $i = 1, \ldots, N+1$, where the index k is such that $T_r \leq t_{i-1} < T_{r+1}$. The presence of the conditional statement involving t_{i-1} complicates these constraints.

A convenient way to model the conditional relationship $T_r \leq t_{i-1} \leq T_{r+1}$ in (4.4.2) is as the following summation.

$$Fz_{ij} \leq a_i \sum_{\substack{r=1 \\ T_r \leq t_{i-1} \leq T_{r+1}}}^{D} \left(\sigma(T_r, m_j) + \frac{\sigma(T_{r+1}, m_j) - \sigma(T_r, m_j)}{T_{r+1} - T_r}(t_i - T_r) \right). \tag{4.4.4}$$

Similarly, we replace (4.4.3) by

$$Fz_{ij} \leq a_i \leq \sum_{\substack{r=1 \\ T_r \leq t_{i-1} \leq T_{r+1}}}^{D} \left(\sigma(T_r, m_j) + \frac{\sigma(T_{r+1}, m_j) - \sigma(T_r, m_j)}{T_{r+1} - T_r}(t_{i-1} - T_r) \right). \tag{4.4.5}$$

We note that only a single term in this summation will be active. The resulting constraint is continuous but not smooth, as t_i passes through the breakpoints. However, this nonsmoothness does not appear to cause any problems for the NLP solvers. In Section 4.5, we provide a reformulation of these nonsmooth constraints that employs

integer variables and a finer discretization of T_r.

4.4.2 Evaluation of Integrals

The model involves integrals over the data functions $k(t, m)$ and $k(t, m) \cdot e(t, m)$ in (4.2.3) and (4.2.7). In [Abramson, 2004], these integrals are evaluated by using Simpson's rule, which is consistent with the piecewise cubic spline interpolation of the data. However, Simpson's rule adds nonlinearity and would be difficult to implement in a modeling language. Instead, we add more data points consistent with the cubic spline interpolation as in (4.4.1), and we evaluate the integrals using the trapezoidal rule on the data points

$$(T_r, k(T_r, m_j)) \quad \text{and} \quad (T_r, k(T_r, m_j) \cdot e(T_r, m_j)) \,.$$

We note that adding more data points does not increase the number of variables in the model and greatly reduces the nonlinearity of the constraints, without sacrificing accuracy, as can be seen from Figure 4.3, which shows the cubic spline versus the piecewise linear approximation of $k(t, \text{epoxy-p})$. The additional data points clearly improve the fidelity of the piecewise linear approximation, as can be seen from the dotted line that represents a piecewise linear interpolation of the original data points.

In general, the temperatures at the intercepts, t_i, will not take the values used in the discretization, T_r, which we must take into account when calculating the values of the integrals. Figure 4.4 illustrates our approach. For a given integration range $[t_{i-1}, t_i]$, we split the integral into three distinct areas (A, B, and D in Figure 4.4) depending on the relative position of the variables t_{i-1}, t_i and the discretization points

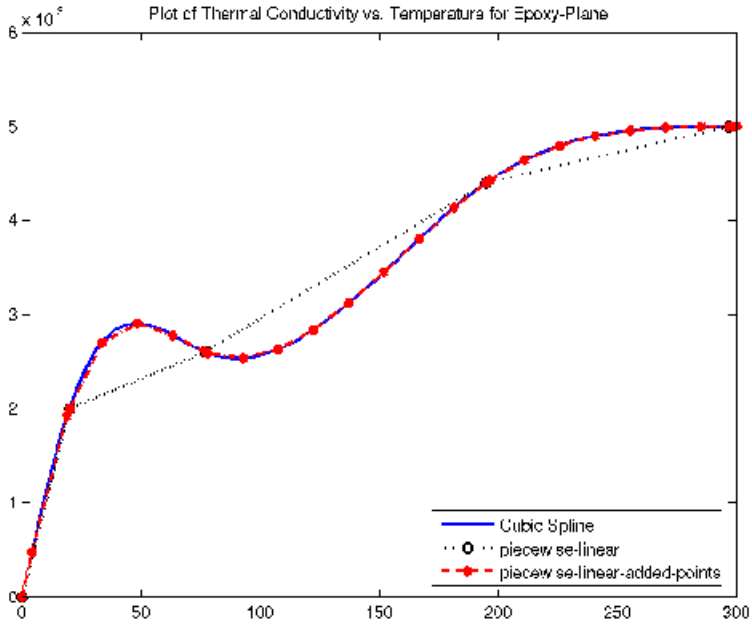

Figure 4.3: Cubic spline versus piecewise linear approximation of $k(t, \text{epoxy-p})$.

T_r. This partition leads to the following disjoint index sets:

$$\begin{aligned}
\mathcal{B} &:= \{1 \le r \le D-1 : [T_r, T_{r+1}] \subset [t_{i-1}, t_i]\} && (B)\\
\mathcal{A} &:= \{1 \le r \le D-1 : t_{i-1} \in (T_r, T_{r+1}) \text{ and } t_i \ge T_{r+1}\} && (A)\\
\mathcal{D} &:= \{1 \le r \le D-1 : t_i \in (T_r, T_{r+1}) \text{ and } t_{i-1} \le T_r\} && (D)\\
\mathcal{E} &:= \{1 \le r \le D-1 : (T_r, T_{r+1}) \supset [t_{i-1}, t_i]\} && (E),
\end{aligned} \tag{4.4.6}$$

where the set $\mathcal{E}$ corresponds to the case where $[t_{i-1}, t_i]$ falls entirely into a single discretization interval. In each case, we approximate the integrals using the trapezoidal rule. We note that in case (B), we can precompute this approximation as parameters

$$K_{r,j} \approx \int_{t=T_r}^{T_{r+1}} k(t, m_j)dt \text{ and } E_{r,j} \approx \int_{t=T_r}^{T_{r+1}} k(t, m_j)e(t, m_j)dt$$

for each interval $[T_r, T_{r+1}]$ and for all materials m_j.

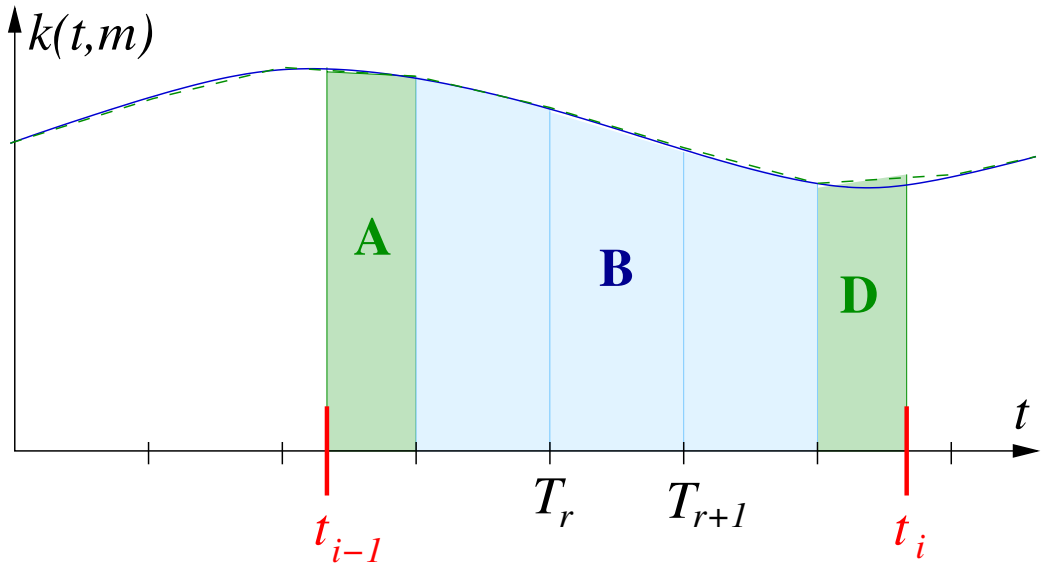

Figure 4.4: Illustration of the integral computation.

Next, we introduce notation to denote the trapezoidal approximations in cases (A), (D), and (E). For a function $g(t, m_j)$ we denote

$$A(g, m_j, r, t_{i-1}) := \frac{1}{2}\left(g(T_{r-1}, m_j) + \frac{g(T_r, m_j) - g(T_{r-1}, m_j)}{T_r - T_{r-1}}(t_{i-1} - T_{r-1}) + g(T_r, m_j)\right)(T_r - t_{i-1}),$$

$$D(g, m_j, r, t_i) := \frac{1}{2}\left(g(T_{r+1}, m_j) + \frac{g(T_r, m_j) - g(T_{r+1}, m_j)}{T_r - T_{r+1}}(T_{r+1} - t_i) + g(T_r, m_j)\right)(T_{r+1} - t_i),$$

and

$$E(g, m_j, r, t_i) := \tfrac{1}{2}\left(g(T_r, m_j) + \tfrac{g(T_{r+1}, m_j) - g(T_r, m_j)}{T_{r+1} - T_r}(t_{i-1} - T_r) + g(T_r, m_j) + \tfrac{g(T_{r+1}, m_j) - g(T_r, m_j)}{T_{r+1} - T_r}(t_i - T_r) + g(T_r, m_j)\right)(T_i - t_{i-1})$$

as the trapezoidal approximation in the areas identified by the sets $\mathcal{A}$, $\mathcal{D}$, and $\mathcal{E}$, respectively. Introducing variables v_{ij} and w_{ij} that approximate

$$v_{ij} \approx \int_{t_{i-1}}^{t_i} k(t, m_j)dt, \quad w_{ij} \approx \int_{t_{i-1}}^{t_i} k(t, m_j)e(t, m_j)dt,$$

we can express v_{ij} and w_{ij} as

$$v_{ij} = \sum_{r\in\mathcal{B}} K_{r,j} + \sum_{r\in\mathcal{A}} A(k, m_j, r, t_{i-1}) + \sum_{r\in\mathcal{D}} D(k, m_j, r, t_i) + \sum_{r\in\mathcal{E}} E(k, m_j, r, t_i) \quad (4.4.7)$$

and

$$w_{ij} = \sum_{r\in\mathcal{B}} E_{r,j} + \sum_{r\in\mathcal{A}} A(k\cdot e, m_j, r, t_{i-1}) + \sum_{r\in\mathcal{D}} D(k\cdot e, m_j, r, t_i) + \sum_{r\in\mathcal{E}} E(k\cdot e, m_j, r, t_i). \quad (4.4.8)$$

The introduction of v_{ij} and w_{ij} allows us to formulate the load-bearing thermal insulation design MVP as a standard MINLP. Before proceeding, however, we simplify some of the nonlinear expressions further to avoid division by zero that can confound standard NLP software. Thus, we rewrite (4.2.3) as

$$q_i x_i = a_i \sum_{j=1}^{|\mathcal{M}|} z_{ij} v_{ij}, \quad i = 1, \ldots, N+1. \quad (4.4.9)$$

We introduce a new variable $u_i = \Delta x_i / x_i$ to model the thermal expansion of each layer, and we rewrite (4.2.7) as

$$u_i \sum_{j=1}^{|\mathcal{M}|} z_{ij} v_{ij} = \sum_{j=1}^{|\mathcal{M}|} z_{ij} w_{ij}, \quad i = 1, \ldots, N+1. \quad (4.4.10)$$

Thus, we can write (4.3.30) as

$$\sum_{i=1}^{N+1} u_i x_i \leq L\frac{\delta}{100}. \tag{4.4.11}$$

This reformulation removes the categorical variables $m \in \mathcal{M}$ from the model and leaves us with a standard MINLP given as follows.

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{N+1} q_i y_i \left(C(t_{i-1})(\frac{T_H}{t_{i-1}} - 1) - C(t_i)(\frac{T_H}{t_i} - 1) \right) \ := \ CoolingPower \\
\text{subject to} \quad & (4.3.13), (4.3.14), (4.3.16), (4.3.25), (4.3.32), (4.3.28), (4.3.33) \\
& (4.3.37), (4.3.35), (4.3.38), (4.4.9), (4.4.10), (4.4.11) \\
& (4.4.4), (4.4.7), (4.4.8), \quad \forall j = 1, \ldots, |\mathcal{M}|, \ i = 1, \ldots, N+1 \\
& (4.4.5), \quad \forall m_j \in \{\text{epoxy-n}, \text{epoxy-p}\}, \ i = 1, \ldots, N+1 \qquad \text{(P-0)} \\
& t_0 = T_C \quad \text{and} \quad t_{N+1} = T_H \\
& y_i \in \{0, 1\}, \quad \forall i = 1, \ldots, N+1 \\
& z_{ij} \in \{0, 1\}, \quad \forall i = 1, \ldots, N+1, \ j = 1, \ldots, |\mathcal{M}| \\
& x_i, q_i, u_i, v_{ij}, w_{ij}, n, t_i, a_i \in \mathbb{R}
\end{aligned}$$

We note that this model contains discontinuous objective coefficients. Next, we show how to reformulate these discontinuous objective coefficients to arrive at a smoother MINLP.

4.4.3 Modeling a Discontinuous Function with Binary Variables

The discontinuous thermodynamic cycle efficiency coefficient $C(t_i)$ in (4.2.1) can be replaced by a smooth relationship. We start by introducing additional binary variables $s_{ki} \in \{0,1\},\ k = 1,\ldots,3$ to model the following implications.

$$t_i \leq 4.2 \Rightarrow\ s_{1i} = 1 \tag{4.4.12}$$

$$4.2 < t_i < 71 \Rightarrow\ s_{2i} = 1 \tag{4.4.13}$$

$$t_i \geq 71 \Rightarrow\ s_{3i} = 1 \tag{4.4.14}$$

Letting $\epsilon > 0$ be a small constant that models the strict inequalities in (4.2.1), we can model the implications (4.4.12) and (4.4.14) as

$$t_i - (T_C - 4.2 - \epsilon)s_{1i} \ \geq\ 4.2 + \epsilon \tag{4.4.15}$$

$$t_i - (T_H - 71 + \epsilon)s_{3i} \ \leq\ 71 - \epsilon, \tag{4.4.16}$$

respectively. Condition (4.4.13) is equivalent to

$$s_{2i} = 0\ \Rightarrow\ t_i \leq 4.2\ \text{ or }\ t_i \geq 71,$$

which we model as follows:

$$s_{1i} + s_{2i} + s_{3i} \ =\ 1 \tag{4.4.17}$$

$$t_i + (T_H - 4.2)s_{1i} \ \leq\ T_H \tag{4.4.18}$$

$$t_i - (71 - T_C)s_{i3} \ \geq\ T_C. \tag{4.4.19}$$

Equation (4.4.17) models the implication that each t_i is in exactly one interval. The two inequalities (4.4.18) and (4.4.19) fix $s_{ik} \in \{0,1\}$ given any t_i, and constrain t_i for any valid choice of $s_{ik} \in \{0,1\}$. The approach presented here is fairly general and applies to other piecewise functions as well.

4.4.4 A Piecewise Smooth MINLP Model

The smooth MINLP model is now given as follows.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{N+1} q_i y_i \left((5s_{1,i-1} + 4s_{2,i-1} + 2.5s_{3,i-1}) \left(\frac{T_H}{t_{i-1}} - 1 \right) \right. \\
& \left. -(5s_{1i} + 4s_{2i} + 2.5s_{3i}) \left(\frac{T_H}{t_i} - 1 \right) \right) \; := \; CoolingPower \\
\text{subject to} \quad & (4.3.13), (4.3.14), (4.3.16), (4.3.25), (4.3.32), (4.3.28), (4.3.33) \\
& (4.3.37), (4.3.35), (4.3.38), (4.4.9), (4.4.10), (4.4.11) \\
& (4.4.7), (4.4.8), (4.4.4), \quad \forall j = 1, \ldots, |\mathcal{M}|, \; i = 1, \ldots, N+1 \\
& (4.4.5), \quad \forall m_j \in \{\text{epoxy-n}, \text{epoxy-p}\}, \; i = 1, \ldots, N+1 \qquad \text{(P-1)} \\
& (4.4.15), (4.4.16), (4.4.17), (4.4.18), (4.4.19), \quad \forall i = 1, \ldots, N+1 \\
& t_0 = T_C \quad \text{and} \quad t_{N+1} = T_H \\
& y_i \in \{0,1\}, \quad \forall i = 1, \ldots, N+1 \\
& z_{ij} \in \{0,1\}, \quad \forall i = 1, \ldots, N+1, \; j = 1, \ldots, |\mathcal{M}|, \\
& s_{ki} \in \{0,1\}, \quad \forall i = 1, \ldots, N+1, \; k = 1, \ldots, 3 \\
& x_i, q_i, u_i, v_{ij}, w_{ij}, n, t_i, a_i \in \mathbb{R}
\end{aligned}
$$

When we ran this model, we noticed that the objective function $CoolingPower$ can become negative, which is a nonphysical solution. More specifically, solving a

relaxation of (P-1) at a node in a branch-and-bound procedure can yield a negative solution. The reason for this behavior is that the variables s_{ki} now have fractional values, and thus the differences

$$\left((5s_{1,i-1} + 4s_{2,i-1} + 2.5s_{3,i-1})\left(\frac{T_H}{t_{i-1}} - 1\right)\right.$$
$$\left.-(5s_{1i} + 4s_{2i} + 2.5s_{3i})\left(\frac{T_H}{t_i} - 1\right)\right) \quad \forall i = 1, \ldots, N+1$$

need not be non-negative. Thus we add the constraint $CoolingPower >= 0$ to the models.

We note that model (P-1) is nonsmooth as a result of the presence of conditional statements in constraints (4.4.4), (4.4.5), (4.4.7), and (4.4.8), which include summations that depend on the temperature t_i. These nonsmooth constraints may cause trouble for standard NLP solvers. We show in Section 4.6 that we can solve such models despite the presence of nonsmooth equations.

4.5 A Smooth MINLP Model with Discretized Temperature

In this section we describe an alternative formulation of the thermal insulation model that assumes that we select the temperatures at the intercepts, t_i, from a discrete set (4.4.1). This model is an approximate formulation, because the temperatures are chosen from a finite set. This formulation may at first sight seem more complicated, but it allows us to remove the nonsmoothness present in model (P-1). We envisage using a large number of discretization points (e.g., at one-degree level). This assumption may seem strong. However, we note that we can always use the discrete values

of t_i to discover the optimal value of the structure variables, n and $m_i \in \mathcal{M}$ and then run an NLP to adjust the temperatures and thickness of the final design. This simplification is justified by the fact that the solution of a MINLP is typically more sensitive to the choice of the integer variables than to the choice of the continuous variables.

As before, we model the discrete choice of temperatures by introducing SOS-1 variables,

$$t_i = \sum_{r=1}^{D} d_{ir} T_r, \quad 1 = \sum_{r=1}^{D} d_{ir}, \quad d_{ir} \in \{0,1\}, \quad \forall i = 0, \ldots, N+1, \tag{4.5.1}$$

where $d_{ir} = 1$ if intercept i is kept at temperature T_r, and 0 otherwise. We note that the new variables d_{ir} are a SOS-1, and so we need to go only to at most $\log(D)$ branching levels in the tree, making this an efficient reformulation.

Introducing a discrete set of temperatures simplifies the MINLP model in a number of ways. We start by simplifying constraints (4.4.7) and (4.4.8). We precompute the integrals of $k(t, m_j)$ and $k(t, m_j)e(t, m_j)$ over $[T_1, T_r]$ as fixed parameters:

$$V_{rj} \approx \int_{T_1}^{T_r} k(t, m_j)dt \quad \text{and} \quad W_{rj} \approx \int_{T_1}^{T_r} k(t, m_j)e(t, m_j)dt$$

for all $r = 1, \ldots, D$ and $j = 1, \ldots, |\mathcal{M}|$. Next, we use the identity

$$v_{ij} = \int_{t=t_{i-1}}^{t_i} k(t, m_j)dt = \int_{t=T_1}^{t_i} k(t, m_j)dt - \int_{t=T_1}^{t_{i-1}} k(t, m_j)dt$$

and observe that (4.5.1) implies

$$\int_{t=T_1}^{t_i} k(t, m_j)dt = \sum_{r=1}^{D} d_{ir} \int_{t=T_1}^{T_r} k(t, m_j)dt.$$

Thus, equations (4.4.7) and (4.4.8) simplify to the following set of *linear* equations:

$$v_{ij} = \sum_{r=1}^{D} d_{ir} V_{rj} - \sum_{r=1}^{D} d_{i-1,r} V_{rj} \quad \forall i = 0, \ldots, N,\ j = 1, \ldots, |\mathcal{M}|, \tag{4.5.2}$$

$$w_{ij} = \sum_{r=1}^{D} d_{ir} W_{rj} - \sum_{r=1}^{D} d_{i-1,r} W_{rj} \quad \forall i = 0, \ldots, N,\ j = 1, \ldots, |\mathcal{M}|. \tag{4.5.3}$$

Next, we show how (4.5.1) simplifies the objective function. First, we observe that

$$\left(\frac{T_H}{t_i} - 1\right) \;=\; \left(\frac{T_H}{\sum_{r=1}^{D} d_{ir} T_r} - 1\right) \;=\; \left(\sum_{r=1}^{D} d_{ir} \frac{T_H}{T_r} - 1\right). \tag{4.5.4}$$

We can replace the binary variables s_{ik} modeling the discontinuous objective coefficients by defining constants

$$\hat{C}_r := \begin{cases} 5 & \text{if } T_r \leq 4.2 \\ 4 & \text{if } 4.2 < T_r < 71\ , \\ 2.5 & \text{if } T_r \geq 71 \end{cases} \tag{4.5.5}$$

which replace $C(t_i)$ in the objective function. Combining (4.5.4) and (4.5.5), we reformulate the objective function as

$$\sum_{i=1}^{N+1} q_i \left(\sum_{r=1}^{D} (d_{i-1,r} - d_{ir}) \hat{C}_r \left(\frac{T_H}{T_r} - 1 \right) \right) \;:=\; \mathit{CoolingPower}.$$

The constraints that bound the temperature difference between consecutive layers, (4.3.35), can be expressed as binary inequalities between d_{ir} and $d_{i-1,r}$:

$$\sum_{r=1}^{D} r d_{ir} \geq \sum_{r=1}^{D} r d_{i-1,r} + y_i, \quad \forall i = 1, \ldots, N+1, \tag{4.5.6}$$

which implies that $t_i > t_{i-1}$ by at least one discretization difference. Similarly, we can express constraints (4.3.37) as a discrete set of constraints:

$$d_{iD} \geq (1 - y_{i+1}), \quad \forall i = 1, \ldots, N. \tag{4.5.7}$$

We next simplify the area stress constraints (4.4.4) and (4.4.5). We start by rewriting the bilevel constraint (4.2.5) as an infinite set of constraints:

$$F \leq a_i \sigma(t, m_i), \quad \forall t \in [t_{i-1}, t_i], \quad \forall i = 1, \ldots, N+1.$$

We observe that the discrete range of the temperature variable allows us to formulate this constraint as a finite set of inequalities:

$$F \leq a_i \sigma(T_s, m_i), \quad \forall s : t_{i-1} \leq T_s \leq t_i, \quad \forall i = 1, \ldots, N+1.$$

We introduce the tensile yield strength $\sigma_{sj} := \sigma(T_s, m_j)$ of each material at all discrete temperature values to remove the dependence on the categorical variable m_i:

$$F \leq a_i \sum_{j=1}^{|\mathcal{M}|} z_{ij} \sigma_{sj}, \quad \forall s : t_{i-1} \leq T_s \leq t_i, \quad \forall i = 1, \ldots, N+1.$$

We formulate the condition $s : t_{i-1} \leq T_s \leq t_i$ using the SOS variables d_{ir} as follows. First, we introduce the following partial sums for notational convenience:

$$p_{is} := \sum_{r=1}^{s} d_{i-1,r} - \sum_{r=1}^{s-1} d_{ir} \; = \; \begin{cases} 1 & \text{if } s : t_{i-1} \leq T_s \leq t_i \\ 0 & \text{else,} \end{cases}$$

which follows from (4.5.1). Thus, we can write (4.2.5) as

$$F \leq a_i \sum_{j=1}^{|\mathcal{M}|} z_{ij}\sigma_{sj} + (1-p_{is})F_{\max} \quad \forall s = 1,\ldots,D-1 \quad \forall i = 1,\ldots,N+1, \quad (4.5.8)$$

where $F_{\max}$ is an upper bound on F.

We can further tighten the formulation by modeling the implications

$$d_{ir} = 1 \Rightarrow \sum_{s=r}^{D} d_{j,s} = 0 \quad \forall j = 1\ldots i-1,\ i = 1\ldots N+1,\ r = 1\ldots D$$
$$d_{ir} = 1 \Rightarrow \sum_{s=1}^{r} d_{j,s} = 0 \quad \forall j = i+1\ldots N+1,\ i = 1\ldots N+1,\ r = 1\ldots D,$$

with the valid inequalities

$$\sum_{j=1}^{i-1}\sum_{s=r}^{D} d_{js} + \sum_{j=i+1}^{N+1}\sum_{s=1}^{r} d_{j,s} \leq B(1-d_{ir}) \quad \forall i = 1\ldots N+1,\ r = 1\ldots D, \quad (4.5.9)$$

where $B \equiv 2N$ is an upper bound on the left-hand side of the inequality.

The resulting model is a smooth MINLP, to which standard MINLP solution techniques can now be applied to find a local solution.

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{N+1} q_i \left(\sum_{r=1}^{D} (d_{i-1,r} - d_{ir}) \hat{C}_r \left(\frac{T_H}{T_r} - 1 \right) \right) := CoolingPower \\
\text{subject to} \quad & (4.3.13), (4.3.14), (4.3.16), (4.3.25), (4.3.32), (4.3.28), (4.3.33) \\
& (4.3.38), (4.4.9), (4.4.10), (4.4.11) \\
& (4.5.1), (4.5.2), (4.5.3) \\
& (4.5.6), (4.5.7), (4.5.8), (4.5.9) \qquad \text{(P-2)} \\
& t_0 = T_C \quad \text{and} \quad t_{N+1} = T_H
\end{aligned}$$

$$y_i \in \{0,1\}, \quad \forall i = 1,\ldots,N+1$$

$$z_{ij} \in \{0,1\}, \quad \forall i = 1,\ldots,N+1,\ j = 1,\ldots,|\mathcal{M}|,$$

$$d_{ir} \in \{0,1\}, \quad \forall i = 1,\ldots,N+1,\ r = 1,\ldots,D,$$

$$x_i, q_i, u_i, v_{ij}, w_{ij}, n, t_i, a_i \in \mathbb{R}$$

Model (P-2) has fewer nonlinear constraints than either (P-0) or (P-1) and is a smooth MINLP. On the other hand, the additional SOS-1 variables d_{ir} make the model larger.

4.6 Solution Methodology and Computational Results

We have experimented with three models with different smoothness properties. All models are available in AMPL [Fourer et al., 1993] from the author upon request. The main characteristics of the three models for $N = 10$ are summarized in Table 4.4. The problem size is roughly linear in N, and the model with $N = 20$ has about twice as many variables and constraints as the model with $N = 10$.

Table 4.4: Comparison of MINLP Models for $N = 10$

Model	# Variables (bin/SOS/int/cont)	# Constraints (nonlinear/linear)	Smoothness
MVP			discontinuous objective & constraints
(P-0)	144 (88/0/1/45)	212 (115/97)	discontinuous objective
(P-1)	174 (118/0/1/55)	262 (115/147)	nonsmooth constraints
(P-2)	3696 (3596/11/1/99)	6715 (3281/3434)	smooth model

We employ standard solution techniques for the solution of the MINLP models

(P-0), (P-1), and (P-2) such as branch-and-bound method [Dakin, 1965, Gupta and Ravindran, 1985], and the LP/NLP-based branch-and-bound method [Quesada and Grossmann, 1992]; see Grossmann [2002] for a recent survey of solution techniques for MINLP problems.

We note that the models (P-0), (P-1), and (P-2) are nonconvex, making it hard to solve these models to global optimality. We employ a standard nonlinear branch-and-bound-based solver, MINLP-BB [Leyffer, 1998], for solving these models. We resort to the strategy of multistarts for solving the nonconvex models (P-0) and (P-1). The strategy of multistarts involves solving a problem at a node a number of times from various starting points. For every node of the branch-and-bound tree, we store a logical switch that indicates whether to perform multiple starts. Initially, this switch is set to true, and we perform $R(= 5)$ restarts from randomly generated starting points and select the best solution value obtained as the solution of that node. If all runs give the same solution value, then we set the logical switch to false for this node and for all its children. Otherwise, the switch remains true for all child nodes. Thus, for convex problems, we perform a multistart only at the root node.

We also set priorities on the integer variables in the model for MINLP-BB to perform branching. For the model (P-0), the integer variable n is given the highest priority, so that the solver branches on a fractional value of the variable n, before branching on other variables. The variables y_i have the next highest priority, since they are dependent on n. The variables for the choice of materials, z_{ik} are accorded the next highest priority, since they are dependent on both n and y_i. For the model (P-1), the variables for the discontinuous objective coefficients, s_{ik} are given the least priority.

We ran these models on a machine with 64-bit AMD Opteron microprocessors

Table 4.5: Results for runs for model (P-0)

	Results for $N = 10$				Results for $N = 20$			
	# intercepts n: 10				# intercepts n: 20			
	Objective value f^*: 1.039				Objective value f^*: 0.988			
	# nodes : 3979				# nodes : 13515			
	# QPs solved : 1559347				# QPs solved : 6229783			
	Time taken (in seconds) : 29894.38				Time taken (in seconds) : 283034.29			
i	t_i	x_i	a_i	z_{ik}	t_i	x_i	a_i	z_{ik}
0	4.2	.	.	.	4.2	.	.	.
1	4.21	1.00	3.102	EPOXYP	4.21	1.00	3.102	EPOXYN
2	7.78	8.01	3.102	EPOXYN	5.78	3.99	3.102	EPOXYN
3	13.58	9.45	3.101	EPOXYN	7.92	4.29	3.102	EPOXYN
4	22.69	11.15	3.100	EPOXYN	10.53	4.74	3.101	EPOXYN
5	39.24	13.99	3.093	EPOXYN	13.91	5.17	3.101	EPOXYN
6	70.88	16.01	3.029	EPOXYN	18.19	5.54	3.100	EPOXYN
7	71	1.05	2.913	EPOXYP	23.68	6.23	3.099	EPOXYN
8	122.21	13.003	3.101	EPOXYN	31.34	7.29	3.091	EPOXYN
9	174.15	8.807	3.734	EPOXYN	41.05	7.17	3.060	EPOXYN
10	179.26	1.00	3.809	EPOXYP	54.66	7.97	3.022	EPOXYN
11	300	16.522	4.474	EPOXYN	70.99	7.66	2.964	EPOXYN
12	.	.	.	.	71	1.00	2.912	EPOXYP
13	.	.	.	.	91.63	5.42	2.928	EPOXYN
14	.	.	.	.	120.99	5.95	3.091	EPOXYN
15	.	.	.	.	146.8	4.54	3.363	EPOXYN
16	.	.	.	.	173.31	4.40	3.721	EPOXYN
17	.	.	.	.	204.29	4.86	4.133	EPOXYN
18	.	.	.	.	228.38	3.57	4.340	EPOXYN
19	.	.	.	.	261.37	4.50	4.472	EPOXYN
20	.	.	.	.	291.3	3.61	4.478	EPOXYN
21	.	.	.	.	300	1.00	4.478	EPOXYN

with a clockspeed of 1.8 GHz, 2 GB RAM, running the Fedora Core 2 operating system. The models were run without any time limits, with the aim of letting the branch-and-bound enumeration run to completion. For the model (P-0), we did two different runs, choosing the parameter N, the maximum number of intercepts in the system, to be 10 and 20. The results of the runs for $N = 10$ and $N = 20$ are summarized in Table 4.5. By increasing the number of intercepts, we were able to improve the solution obtained by Abramson [2004] from 1.0623 to 1.039 and 0.988 for $N = 10$ and $N = 20$, respectively.

We note that solving the model with $N = 20$ intercepts is computationally intensive. In order to solve these models faster, we solve the models with the number of intercepts n fixed at $n = 1, \ldots, 29$. This allows us to also fix the indicator variables y_i, reducing the number of variables and the solution time significantly. Figure 4.5 shows the value of objective function f as a function of the number of intercepts, n. We see that the objective function value first drops sharply as n is increased and then

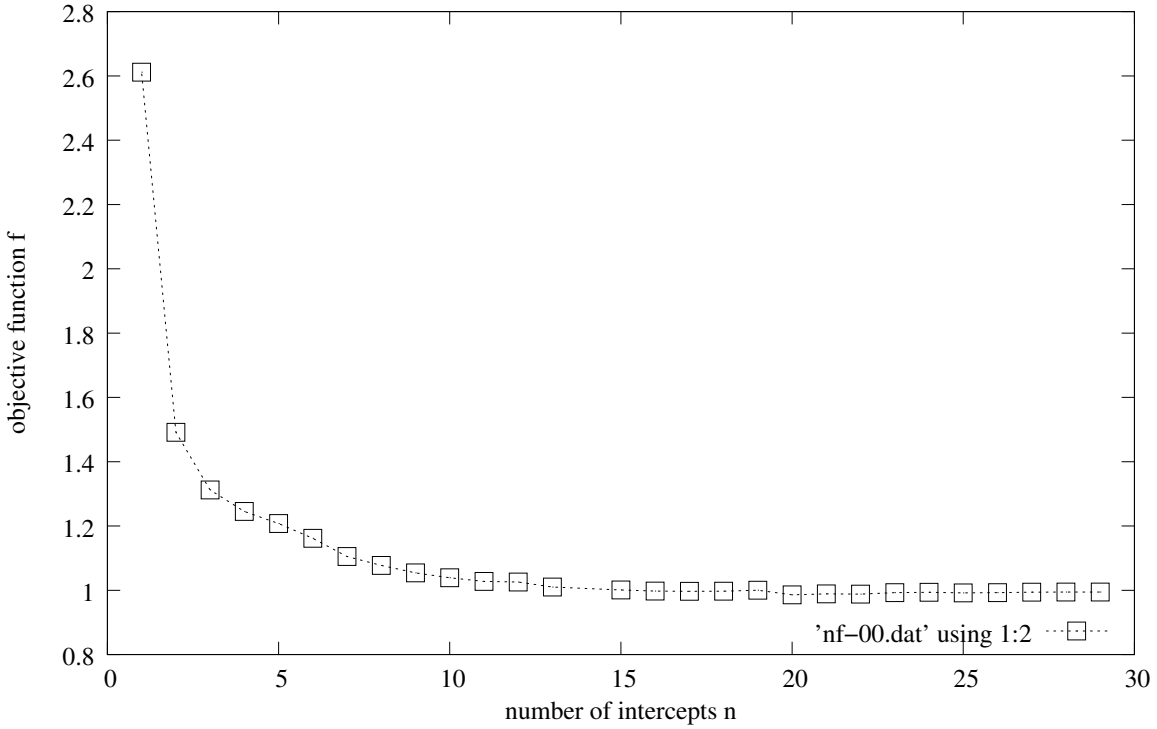

Figure 4.5: Plot showing objective function value for (P-0) with n.

stabilizes as the value of n becomes larger. Clearly, $n = 10$ is not the optimal number of intercepts.

We performed a similar set of experiments for the model P-1. The results of the runs for $N = 10$ and the best solution found by our runs are summarized in Table 4.6. The solution obtained by solving the MINLP models (P-0) and (P-1) are similar to or better than the ones obtained by Abramson [2004]. Also, by increasing the number

of intercepts in the model, we are able to obtain better solutions than the model with at most 10 intercepts. The MINLP model formulated makes it possible to search in the larger sub-space of all the possible materials at an intercept. Thus, we are able to obtain solutions where different materials are used as insulators in different layers in the same configuration. We see that the material chosen in the final configuration is either epoxy(normal) or epoxy(plane).

Table 4.6: Results for runs for model (P-1)

	Results for $N = 10$				Best Solution Found for (P-1)			
	# intercepts n: 10				# intercepts n: 21			
	Objective value f^*: 1.02165				Objective value f^*: 0.98305			
	# nodes : 382540				# nodes : 244985			
	# QPs solved : 7049146				# QPs solved : 9360693			
	Time taken (in seconds) : 28778.95				Time taken (in seconds) : 74283.38			
i	t_i	x_i	a_i	z_{ik}	t_i	x_i	a_i	z_{ik}
0	4.2	.	.	.	4.2	.	.	.
1	4.21	1.00	3.102	EPOXYN	4.21	1.00	3.102	EPOXYN
2	7.78	8.19	3.102	EPOXYN	5.71	3.75	3.102	EPOXYN
3	13.58	9.66	3.101	EPOXYN	7.61	4.07	3.102	EPOXYN
4	22.66	11.41	3.100	EPOXYN	10.03	4.41	3.101	EPOXYN
5	39.17	14.32	3.093	EPOXYN	13.05	4.78	3.101	EPOXYN
6	70.99	16.49	3.029	EPOXYN	16.79	5.17	3.101	EPOXYN
7	71	1.00	2.913	EPOXYP	21.43	5.62	3.000	EPOXYN
8	112.92	9.73	3.034	EPOXYN	27.41	6.19	3.096	EPOXYN
9	154.38	7.37	3.461	EPOXYN	34.57	6.13	3.076	EPOXYN
10	201.53	7.66	4.108	EPOXYP	43.12	6.09	3.048	EPOXYN
11	300	13.16	4.474	EPOXYN	55.46	7.06	3.014	EPOXYN
12	.	.	.	.	70.99	6.78	2.961	EPOXYN
13	.	.	.	.	71	1.00	2.912	EPOXYN
14	.	.	.	.	91.10	5.21	2.927	EPOXYN
15	.	.	.	.	109.95	3.96	3.013	EPOXYN
16	.	.	.	.	128.48	3.44	3.162	EPOXYN
17	.	.	.	.	147.06	3.21	3.367	EPOXYN
18	.	.	.	.	165.84	3.12	3.616	EPOXYN
19	.	.	.	.	184.96	3.09	3.890	EPOXYN
20	.	.	.	.	206.75	3.40	4.157	EPOXYN
21	.	.	.	.	233.52	3.95	4.371	EPOXYN
22	.	.	.	.	300	8.50	4.475	EPOXYN

The addition of new binary variables s_{ki} to model the discontinuous objective

function coefficients makes the NLP easier to solve in terms of the number of QP's that are needed to solve per NLP. Table 4.7 shows the average number of QP's solved per NLP for the models (P-0) and (P-1). We observe that this number remains almost constant for (P-1), while it increases in a near-linear fashion for (P-0). For the largest models we observe an order of magnitude reduction in the number of QP's solved per NLP. We believe that this effect is due to the lack of continuity in model (P-0).

Table 4.7: Results showing the average number of QP's solved per NLP for (P-0) and (P-1)

n	(P-0)	(P-1)	n	(P-0)	(P-1)
1	26.6	25.15	9	388.36	29.34
2	49.77	33.20	10	407.48	30.26
3	124.77	34.70	19	512.43	32.30
4	209.97	42.25	21	522.03	38.20
5	247.01	35.16	23	528.14	41.05
6	300.07	33.21	25	532.64	43.32
7	300.15	29.32	28	544.19	47.26
8	342.64	32.23	.	.	.

The discretized model (P-2) is smooth as a result of the modeling of temperature at intercepts using discrete variables. Even though the nonlinearity in the model has been considerably reduced, it is still nonconvex. It also has a large number of variables and constraints. We attempted to solve this model using MINLP-BB but observed excessive solution times. Hence, we report the solution statistics only for (P-2) with FilMINT, a linearizations-based solver for mixed-integer nonlinear programs described in Chapter 2. It implements the LP/NLP algorithm in a branch-and-cut framework. We refer the reader to Quesada and Grossmann [1992] and Chapter 1 of this thesis for the details of the algorithm. For nonconvex problems, the solution methodology can be seen essentially as a heuristic. FilMINT's advanced MILP features make it possible to obtain good upper bounds quickly which helps the

solution scheme for the problem. For the nonconvex model (P-2), we use the basic version of the LP/NLP algorithm, because linearizations added to the model may cut off the feasible region.

FilMINT is competitive with all other software available for convex MINLPs. In particular, FilMINT often outperforms BONMIN-v2. We note that the performance of Bonmin (version 1) is superior to the commercial packages DICOPT and SBB [Bonami et al., 2006].

We note that the full 1-degree discretization that we do for model (P-2) is not necessary, because we can be reasonably sure that the intercepts close to the cold end will not have high temperatures, and vice versa. We reduce the discretization level of temperature for an intercept i by restricting the intercept to lie in a smaller interval $[T_{l_i}, T_{u_i}]$ than from $[T_1, T_D]$, where $l_i, u_i \in [1, D]$. This implies modeling the SOS-1 variables d_{ir} as

$$t_i = \sum_{r=l_i}^{u_i} d_{ir} T_r, \quad 1 = \sum_{r=l_i}^{u_i} d_{ir}, \quad d_{ir} \in \{0, 1\}, \quad \forall i = 0, \ldots, N+1. \tag{4.6.1}$$

This reduces the number of integer variables in the model and makes the problem easier to solve. This procedure of choosing different temperature intervals $[T_{l_i}, T_{u_i}]$ is essentially a heuristic with great flexibility. We base our discretization reduction strategy around the solution of models (P-0) and (P-1). For $n = 10$, Table 4.8 shows the temperature intervals that we allow for different intercepts as part of our strategy.

We also restrict the choice of materials in the model to epoxy-plane and epoxy-normal. This is a reasonable assumption because we see that the solution obtained for

Table 4.8: Table showing the discretization reduction strategy for (P-2) for $n = 10$

Intercept i	Temperature Range	Intercept i	Temperature Range
1	$[1, 81]$	6	$[55, 135]$
2	$[1, 81]$	7	$[55, 135]$
3	$[1, 81]$	8	$[82, 189]$
4	$[1, 81]$	9	$[82, 189]$
5	$[1, 81]$	10	$[163, 243]$

models (P-0) and (P-1) have only these materials in their final configuration. To better handle the nonconvex constraints, we employ the strategy of adding linearizations as local cuts in some of our runs. Hence, the cuts generated at some node are added only to the subtree rooted at that node, instead of the default FilMINT strategy to adding these linearizations globally on all open nodes. For nonconvex problems, this heuristic helps in reducing the feasible region that can be cut off. The results of the runs for $N = 10$ with the mentioned discretization reduction strategy and of the runs using local cuts are shown in Table 4.9.

We note that the solution obtained in our runs is only a little worse compared to models (P-0) and (P-1). We see that local cuts help in reducing, to a large extent, the feasible region from being cut off, thus letting the branch-and-cut enumeration visit more nodes and find better feasible solutions. We also see that the average number of QP's is still small for the smooth model (P-2) compared to the size of the model. We use the solution of the model (P-2) to obtain the value of the discrete variables n, and $m_i \in \mathcal{M}$, and we rerun an NLP to adjust the values of the structural variables. The results are given in Table 4.10. One can see that the values of the structural variables and the objective value are similar to those found by running model (P-0). See Table 4.5 in this regard. This suggests that the smooth model (P-2) is a good approximation of model (P-0). All AMPL models are available online

Table 4.9: Results for runs for model (P-2) for $N = 10$ using FilMINT

	Best Solution with Global Cuts				Best Solution with Local Cuts			
	# intercepts n: 10				# intercepts n: 10			
	Objective value f^*: 1.281264				Objective value f^*: 1.212189			
	# MILP nodes : 5411				# MILP nodes : 15111			
	# LP's solved : 6279				# LP's solved : 27357			
	# NLP's solved : 5				# NLP's solved : 7102			
	# QP's solved : 158				# QP's solved : 286392			
	Average # QP's per NLP : 31.6				Average # QP's per NLP : 40.32			
	Time taken (in seconds) : 5712.02				Time taken (in seconds) : 72007.67			
i	t_i	x_i	a_i	z_{ik}	t_i	x_i	a_i	z_{ik}
0	4.2	.	.	.	4.2	.	.	.
1	16	25.21	3.106	EPOXYN	6	5.54	3.103	EPOXYN
2	40	21.40	3.104	EPOXYN	7	1.99	3.104	EPOXYN
3	41	1.00	3.029	EPOXYP	8	1.85	3.105	EPOXYN
4	71	15.95	3.025	EPOXYN	9	1.73	3.105	EPOXYN
5	72	1.00	2.908	EPOXYN	15	8.91	3.106	EPOXYN
6	137	12.59	3.248	EPOXYN	67	40.56	3.105	EPOXYN
7	138	1.00	3.259	EPOXYN	72	3.86	2.917	EPOXYN
8	191	7.89	3.976	EPOXYN	124	10.75	3.116	EPOXYN
9	192	1.00	3.989	EPOXYN	160	5.73	3.537	EPOXYN
10	204	1.61	4.136	EPOXYN	174	2.49	3.736	EPOXYP
11	300	11.33	4.483	EPOXYN	300	16.58	4.483	EPOXYN

Table 4.10: Results for model (P-0) after fixing discrete variables using (P-2)

	Results for $n = 10$								
	Objective Solution Value f^*: 1.097219								
	# QPs Solved : 508								
	Time Taken (in seconds): 1.032								
i	t_i	x_i	a_i	z_{ik}	i	t_i	x_i	a_i	z_{ik}
0	4.2	.	.	.	6	28.79	8.83	3.099	EPOXYN
1	4.21	1.01	3.102	EPOXYN	7	41.03	8.95	3.071	EPOXYN
2	6.44	5.09	3.106	EPOXYN	8	71	16.23	3.022	EPOXYN
3	9.59	5.70	3.102	EPOXYN	9	134.2	16.33	3.221	EPOXYN
4	13.91	6.38	3.101	EPOXYN	10	155.48	4.02	3.476	EPOXYP
5	19.75	7.13	3.100	EPOXYN	11	300	20.33	4.475	EPOXYN

at www.mcs.anl.gov/~leyffer/MacMINLP.

4.7 Conclusions

We used mixed integer nonlinear programming techniques to model the load-bearing thermal insulation problem. We used integer variables to model the categorical variables, so that the model now allows continuous relaxations and can be solved by using standard MINLP solution techniques. We developed facet-defining inequalities for a relaxation of this reformulated MINLP model. We evaluated integrals by adding more data points consistent with the cubic interpolation of the data and using the trapezoidal rule. We also avoided the second-level optimization in the problem by introducing a finer piecewise linear approximation of the data and by enforcing the bounds at temperatures for each intercept.

Our reformulations give rise to three models with varying degrees of smoothness, and we comment on the relative merits of the formulations. Our computational results indicated that the MINLP formulations obtain better results than previous results by Abramson [2004]. In particular, by increasing the number of intercepts from 10 to 20, we were able to reduce the cooling power by 4%.

The modeling of mixed variable problems as MINLPs allows us to apply more powerful techniques, such as branch-and-bound, outer approximation, and branch-and-cut, rather than a heuristic search technique. Engineering or modeling insight is included into the MILP model by using priorities on the integer variables or by restricting temperature ranges for the intercepts. We believe that the modeling techniques shown here are very general and can be used as a blueprint for modeling other design problems that have categorical variables.

Chapter 5

Conclusions

5.1 Contributions

This is a computationally oriented thesis, where we have studied a number of algorithmic and implementation issues for developing a better framework and solution techniques for solving convex mixed-integer nonlinear programs.

In Chapter 2, we introduce an algorithmic framework that aims to solve convex MINLPs at a cost which is only a small multiple of the cost of solving a comparable MILP. The LP/NLP branch-and-bound algorithm is deemed to be the best algorithm for this purpose. We discuss how to implement this algorithm in a branch-and-cut framework using existing software components for mixed-integer linear programming and nonlinear programming. Implementing the algorithm in a branch-and-cut framework provides us with a convenient mechanism to explore the impact of various advanced MILP features, such as cutting planes, branching and node selection rules, heuristics and preprocessing. We do carefully constructed experiments to understand

the impact of these features. Our results suggest that MILP based heuristics, branching, and node selection rules have a big impact on the solution scheme. We believe that these areas should be further explored by suitably modifying the scheme for solving MINLPs. Our results for cutting planes (based on special linear structures) and preprocessing are not very promising. This is to be expected because there is not enough special structure in the linear part of the model. Also, linearizations do not generally have the special structure required for these methods to prove effective. However, we note that using the nonlinearity information may result in substantial gains with these methods.

We exploit the convexity property of nonlinear functions to derive linearizations about different points for strengthening the formulation. The goal is to get a tighter approximation of the nonlinear feasible region and the objective function. We believe that the use of linear underestimators is the key to solving convex MINLPs efficiently. However, this entails a classical tradeoff between the effort taken to get these linearizations and the quality of these linearizations. We derive different ways of generating linearizations, some of which involve just calculating gradients and doing function evaluations instead of solving nonlinear programs. We do extensive numerical experiments to determine when, how, and how many linearizations should be generated. Our computational experiments suggest that our scheme of generating and managing linearizations is very effective on a wide range of instances. There is ample scope for deriving a better scheme for generating and managing linearizations, and some ideas for this are mentioned in the chapter. The end result of our efforts is an efficient, flexible and powerful solver, called FilMINT, competitive with all other known software available for this problem class.

In Chapter 3, we use the framework that we have developed to investigate different

heuristic schemes for getting feasible solutions for MINLPs. We take two different approaches for getting feasible solutions, based on the feasibility pump framework. The first approach is an extension of the feasibility pump heuristic for MILPs, that iteratively solves a distance-based NLP and its rounding. We investigate a number of different rounding schemes and come up with a few new schemes. This scheme can be quite fast, since rounding is very inexpensive; however, it can fail for some instances due to cycling issues. The second approach that we follow is based on iteratively solving a distance-based NLP and a distance-based MILP. While solving MILPs iteratively can be potentially expensive, this method does not cycle and is an exact solution scheme for solving convex MINLPs. We concentrate on improving the linear model by using linearizations, reducing the time spent in the MILP phase, and finding a better scheme for improving an incumbent, once found. We then propose and implement another feasibility pump scheme, that is based on integrating NLP solves with the branch-and-cut tree search for MILPs, so that we do not have to solve MILPs iteratively. This is akin to the philosophy of the LP/NLP methodology for MINLPs and has potentially enormous advantages. This method also does not cycle, and hence is an exact approach. However, there are some open issues that need to be dealt with, and these are mentioned in the chapter. Our computational results show the effectiveness of our schemes, especially in a branch-and-cut framework.

In Chapter 4, we illustrate the advantages of using MINLP modeling and solution methodologies for solving some important applications that are traditionally modeled as Mixed Variable Programs. A typical MVP model contains categorical variables that do not allow continuous relaxations, and hence the use of branch-and-bound techniques as its solution scheme. We use integer and nonlinear modeling techniques to model this problem. We also show how to handle some other common bottlenecks

in modeling these problems as MINLPs. These issues include the presence of discontinuities and functions for which gradients are difficult to compute. Using a fine discretization to approximate such functions and modeling techniques, we come up with three different MINLP models with varying degree of smoothness. Our approach allows us to use sophisticated solution techniques for solving these problems, instead of relying on heuristic schemes. We use FilMINT and another nonlinear branch-and-bound based MINLP solver for solving these models. Our numerical results justifies our approach and suggests that significant gains may be made by using these modeling and solution techniques. Our approach serves as a blueprint for modeling many such applications, thereby increasing the domain of applications that may be modeled as MINLPs.

The contributions made in this thesis have led to improvements in the solution framework and techniques for convex MINLPs, and also contributed towards the goal of making mixed-integer nonlinear programming a practical and sophisticated mathematical tool for solving challenging theoretical and real-life applications.

5.2 Future Research

The research presented in this thesis has led to an improvement in the branch-and-cut based solution schemes for convex MINLPs. We now describe some important issues which deserve immediate attention, as well as suggest some ideas that should be explored in future.

5.2.1 Primal Heuristics for MINLP

The research described in Chapter 3 has led to a number of new ideas in the field of primal heuristics for MINLPs. The branch-and-pump heuristic, as described in Chapter 3, integrates the feasibility pump within an LP/NLP-based branch-and-cut framework. It continues with the tree search for the MILP and solves NLP subproblems whenever an integer assignment is found, without having to re-start the tree search. However, each time a distance-based NLP is solved, we would ideally like to change the objective function of the MILP, so that the tree is now searched with respect to the new objective. The lower bounds and fathoming rules may have to be changed to achieve this without hurting the tree search procedure. A solution to this issue would lead to an exact LP/NLP procedure for feasibility pump heuristic for MINLPs, with considerable potential benefits.

Further, rounding schemes that take into account feasibility of the rounded solution may lead to substantial improvements in rounding based feasibility pump schemes. We suggest a rounding scheme that takes into account the feasibility of the solution with respect to the nonlinear feasible region. We call this scheme "directed rounding from nonlinear constraint violation". The directed rounding scheme works as follows: Let $(\bar{x}^*, \bar{y}^*)$ be a fractional solution to the NLP relaxation. For every fractional integer variable, compute the effect of rounding on the constraints, i.e., form

$$\hat{y}_i^0 := (\bar{y}_1^*, \ldots, \bar{y}_{i-1}^*, 0, \bar{y}_{i+1}^*, \ldots, \bar{y}_p^*) \text{ and } \hat{y}_i^1 := (\bar{y}_1^*, \ldots, \bar{y}_{i-1}^*, 1, \bar{y}_{i+1}^*, \ldots, \bar{y}_p^*),$$

and compute the constraint violation due to rounding y_i as:

$$\hat{c}_i^0 := \| \max(0, g(\bar{x}^*, \hat{y}_i^0)) \| \text{ , and } \hat{c}_i^1 := \| \max(0, g(\bar{x}^*, \hat{y}_i^1)) \|,$$

and then choose the rounding so as to "minimize" the violation.

We suggest another scheme that is based on filter methods for nonlinear programming. For every fractional value $\bar{y}$ and its rounding $\hat{y}$, we introduce two measures of "success": the usual potential (or fractionality) $\Delta(y, \hat{y})$, defined in Chapter 3, and the constraint violation of the rounded solution $\| \max(0, g(x, \hat{y})) \|$. We can then "accept points" as long as we can add them to the filter. There are two restoration phases in this scheme: To restore feasibility, we can simply minimize $\| \max(0, g(x, \hat{y})) \|$ for fixed $\hat{y}$. To reduce $\Delta(y, \hat{y})$, we could branch or add cutting planes. The use of a filter would allow a more algorithmic approach to a feasibility pump.

We finally note that some other successful heuristics for MILPs, like RINS and local branching, can also be extended for MINLPs in a similar way as described in Chapter 3.

5.2.2 Cutting Planes for MINLP

While the area of cutting planes for mixed-integer linear programming has been extensively explored, the same cannot be said for mixed-integer nonlinear programming. Please refer to Chapter 1 for a brief description of some of the progress made in this area for MINLPs. Cutting plane methods can potentially be very useful for solving MINLPs and deserve immediate attention.

The success of cutting planes for MILPs and the development of a branch-and-cut framework motivates the investigation of different classes of cuts. We now suggest

an idea for developing cutting planes for a linearization-based algorithm for MINLPs, where encoded in the linear structure is information regarding the nonlinearity in the problem, and bounds for the optimal value of the objective function.

Consider the following Benders cut (discussed in Chapter 2) obtained from combining the objective and constraint linearizations with optimal NLP multipliers μ^k from the solution of (NLP(y^k)):

$$\eta \geq f(x^k, y^k) + (\nabla_y f(x^k, y^k)^T + (\mu^k)^T \nabla_y g(x^k, y^k))^T (y - y^k),$$

where (x^k, y^k) is the solution about which linearizations are taken. This valid inequality is a knapsack constraint with integer variables y and just one continuous variable η and is expressed as

$$\sum_{j \in P} a_j y_j \leq b + \eta, \tag{5.2.1}$$

where

$$a = (\nabla_y f(x^k, y^k)^T + (\mu^k)^T \nabla_y g(x^k, y^k)),$$
$$b = (-f(x^k, y^k) + (\nabla_y f(x^k, y^k)^T + (\mu^k)^T \nabla_y g(x^k, y^k)^T) y^k),$$

and P is the index set of integer variables in (MINLP).

We now present different ways of obtaining cuts using the Benders cut.

- Given an upper bound Z_{UB} on (MP($\mathcal{K}$)), and substituting it for η gives us a

knapsack set

$$T' = \left\{ y \in \mathbb{B}^{|P|} | \sum_{j \in P} a_j y_j \leq b + Z_{\text{UB}} \right\},$$

for which knapsack covers can be derived. See papers by Crowder et al. [1983], and Atamtürk [2004] in this regard.

- Given a valid lower bound Z_{LB} on (MINLP), we can do variable substitutions and use results by Marchand and Wolsey [1999] to generate facets for the following set:

$$Q = \left\{ (y, \hat{\eta}) \in \mathbb{B}^{|P|} \times \mathbb{R}_+ | \sum_{j \in P} a_j y_j \leq b + \eta \right\}.$$

- Further, we can generate MIR inequality for the following set:

$$U = \left\{ (y, \hat{\eta}) \in \mathbb{Z}_+^{|P|} \times \mathbb{R}_+ | \sum_{j \in P} a_j y_j \leq b + \eta \right\},$$

by using lower bounds Z_{LB} and doing variable substitutions. Please see the paper by Marchand and Wolsey [2001] in this regard.

These ideas need to be looked at in more details so as to be able to generate effective cuts in our linearization-based framework. Finally, more research needs to be done on both general purpose cuts and those obtained from relaxations of specific sets for MINLPs.

5.2.3 Issues in the design of a parallel LP/NLP branch-and-cut solver

There are a number of issues and challenges in achieving a scalable parallel implementation of the LP/NLP branch-and-cut algorithm on the computational grid. To the best of our knowledge, only Goux and Leyffer [2003] have attempted a parallel implementation of the nonlinear branch-and-bound algorithm for solving MINLPs. Timely and efficient sharing of useful information among the processors and effective handling of the parallel architecture are some of the issues that need attention when designing a parallel code. Some of the key issues are as follows:

- The computational grid is an inexpensive, readily available, and suitable alternative to massively parallel multiprocessors and dedicated clusters for a platform for enabling the implementation of parallel solvers. However, it presents its own set of challenges, including heterogeneity, dynamic availability, high latency and low communication bandwidth. These issues need to be handled while designing a robust parallel implementation. Fortunately, some of these issues can be taken care of by using Condor, a resource manager (Livny et al. [1997]), and MW (Goux et al. [2000b,a]), a framework for building parallel master-worker applications.

- The LP/NLP branch-and-cut algorithm as presented in Chapter 2, may generate a lot of linearizations. This can create serious contention issues at the master (in a Master-Worker framework) and hinder scalability. Sophisticated cut management and cut sharing techniques need to be developed in this regard. This issue is of greater importance in a linearization-based algorithm for MINLPs when compared to cut generation and sharing in MILPs.

Appendix A

For Tables A.1–A.3, the row-headings are described as follows:

- Instance: Name of the instance.
- Vars: Number of variables, C. Vars: Number of continuous variables, and Bin. Vars: Number of binary variables.
- Cons: Number of constraints, Nl. Cons: Number of nonlinear constraints, and L. Cons: Number of linear constraints.

Table A.1: Summary of Properties for Easy Instances

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
alan	8	4	4	7	0	7
batch	46	22	24	73	1	72
batchdes	19	10	9	19	1	18
csched1	76	13	63	22	0	22
enpro48	153	61	92	214	1	213
enpro56	127	54	73	191	1	190
ex1223	11	7	4	13	4	9
ex1223a	7	3	4	9	4	5

Continued on next page

Continued from previous page

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
ex1223b	7	3	4	9	4	5
ex3	32	24	8	31	5	26
ex4	36	11	25	30	25	5
fac1	22	16	6	18	0	18
fac2	66	54	12	33	0	33
fac3	66	54	12	33	0	33
gbd	4	1	3	4	0	4
m3	26	20	6	43	6	37
meanvarx	35	21	14	44	0	44
optprloc	30	5	25	30	25	5
prob02	6	0	0	8	5	3
prob03	2	0	0	1	1	0
ravem	112	58	53	186	1	185
ste14	11	7	4	13	4	9
ste38	4	2	0	3	1	2
stmiqp1	5	0	5	1	0	1
stmiqp2	4	0	2	3	0	3
stmiqp3	2	0	0	1	0	1
stmiqp4	6	3	3	4	0	4
stmiqp5	7	5	2	13	0	13
sttest1	5	0	5	1	0	1
sttest2	6	0	5	2	0	2
sttest3	13	0	10	10	0	10
sttest4	6	0	2	5	0	5

Continued on next page

Continued from previous page

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
sttest5	10	0	10	11	0	11
sttest6	10	0	10	5	0	5
sttest8	24	0	0	20	0	20
sttestgr1	10	0	0	5	0	5
sttestgr3	20	0	0	20	0	20
sttestph4	3	0	0	10	0	10
synthes1-10	6	3	3	6	2	4
synthes1	6	3	3	6	2	4
synthes2	11	6	5	14	3	11
synthes3	17	9	8	23	4	19
tls2	37	4	31	24	2	22
trimloss2	37	6	31	24	2	22
BNS121612	248	56	192	417	1	416
BNS162010	272	72	200	681	1	680
BNS8125	100	40	60	217	1	216
BSBM8664	132	79	53	371	1	370
BSCH8664	318	265	53	599	1	598
Process20M	65	45	20	100	13	87
Process8CH	71	63	8	110	5	105
Process8C	71	63	8	110	5	105
SLBM42	70	38	32	84	32	52
CLay0203H	90	72	18	132	24	108
CLay0203M	30	12	18	54	24	30
CLay0204H	164	132	32	234	32	202

Continued on next page

Continued from previous page

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
CLay0204M	52	20	32	90	32	58
CLay0303H	99	78	21	150	36	114
CLay0303M	33	12	21	66	36	30
FLay02H	46	42	4	51	2	49
FLay02M	14	10	4	11	2	9
FLay03H	122	110	12	144	3	141
FLay03M	26	14	12	24	3	21
FLay04M	42	18	24	42	4	38
SLay04H	140	116	24	174	0	174
SLay04M	44	20	24	54	0	54
SLay05H	230	190	40	290	0	290
SLay05M	70	30	40	90	0	90
Syn05H	42	37	5	58	3	55
Syn05M02H	104	84	20	151	6	145
Syn05M02M	60	40	20	101	6	95
Syn05M03H	156	126	30	249	9	240
Syn05M03M	90	60	30	174	9	165
Syn05M04H	208	168	40	362	12	350
Syn05M04M	120	80	40	262	12	250
Syn05M	20	15	5	28	3	25
Syn10H	77	67	10	112	6	106
Syn10M02H	194	154	40	294	12	282
Syn10M02M	110	70	40	198	12	186
Syn10M03H	291	231	60	486	18	468

Continued on next page

Continued from previous page

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
Syn10M04H	388	308	80	708	24	684
Syn10M	35	25	10	54	6	48
Syn15H	121	106	15	181	11	170
Syn15M02H	302	242	60	467	22	445
Syn15M03H	453	363	90	768	33	735
Syn15M04H	604	484	120	1114	44	1070
Syn15M	55	40	15	89	11	78
Syn20H	151	131	20	233	14	219
Syn20M02H	382	302	80	606	28	578
Syn20M03H	573	453	120	999	42	957
Syn20M	65	45	20	113	14	99
Syn30H	228	198	30	345	20	325
Syn30M02H	576	456	120	900	40	860
Syn40H	302	262	40	466	28	438
Water0202R	195	188	7	283	0	283
Water0303R	384	370	14	556	0	556
Syn40M04H	1528	1208	320	2904	112	2792

Table A.2: Summary of Properties for Moderate Instances

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
m6	86	56	30	157	12	145
BatchStorageBM9964	205	125	80	647	1	646

Continued on next page

Continued from previous page

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
BatchStorageCH9964	527	447	80	1048	1	1047
BatchStorage10BM10	238	149	89	799	1	798
BatchStorageBM10	238	149	89	799	1	798
SafetynorotationBM	168	108	60	162	6	156
BatchS101006M	278	149	129	1019	1	1018
BatchS121208M	406	203	203	1511	1	1510
FLay05M	62	22	40	65	5	60
CLay0304H	176	140	36	258	48	210
CLay0304M	56	20	36	106	48	58
CLay0305M	85	30	55	155	60	95
RSyn0810M	185	111	74	312	6	306
RSyn0815M	205	126	79	347	11	336
SLay06H	342	282	60	435	0	435
Syn15M03M	255	165	90	537	33	504
Syn30M	100	70	30	167	20	147
SafetyLayout4Conv	182	150	32	228	32	196
SafetyLayoutBM52	110	60	50	125	40	85
FLay04H	234	210	24	282	4	278
RSyn0805M02H	700	552	148	1045	6	1039
RSyn0805M04H	1400	1104	296	2438	12	2426
RSyn0810M02H	790	622	168	1188	12	1176
RSyn0815M02H	898	710	188	1361	22	1339
RSyn0820M02H	978	770	208	1500	28	1472
RSyn0830M02H	1172	924	248	1794	40	1754

Continued on next page

Continued from previous page

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
RSyn0840M02H	1360	1072	288	2106	56	2050
SLay06M	102	42	60	135	0	135
Syn10M03M	165	105	60	342	18	324
Syn10M04M	220	140	80	516	24	492
Syn15M02M	170	110	60	313	22	291
Syn20M04H	764	604	160	1452	56	1396
Syn30M03H	864	684	180	1485	60	1425
Syn30M04H	1152	912	240	2160	80	2080
Syn40M02H	764	604	160	1212	56	1156
Syn40M03H	1146	906	240	1998	84	1914
RSyn0810M04H	1580	1244	336	2784	24	2760

Table A.3: Summary of Properties for Hard Instances

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
csched2	400	92	308	137	0	137
fo7	114	72	42	211	14	197
fo72	114	72	42	211	14	197
fo8	146	90	56	273	16	257
fo9	182	110	72	343	18	325
m7	114	72	42	211	14	197
o7	114	72	42	211	14	197
o72	114	72	42	211	14	197

Continued on next page

Continued from previous page

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
tls12	812	144	656	384	12	372
tls4	105	16	85	64	4	60
tls5	161	25	131	90	5	85
tls6	215	36	173	120	6	114
tls7	345	49	289	154	7	147
trimloss12	812	156	656	384	12	372
trimloss4	105	20	85	64	4	60
trimloss5	161	30	131	90	5	85
trimloss6	215	42	173	120	6	114
trimloss7	345	56	289	154	7	147
safetyrotationCH	408	348	60	462	6	456
BatchS151208M	445	242	203	1781	1	1780
BatchS201210M	558	307	251	2327	1	2326
CLay0305H	275	220	55	395	60	335
FLay05H	382	342	40	465	5	460
FLay06H	566	506	60	693	6	687
RSyn0805H	720	424	296	1886	12	1874
RSyn0805M02M	360	212	148	769	6	763
RSyn0805M03M	540	318	222	1284	9	1275
RSyn0805M04M	720	424	296	1886	12	1874
RSyn0810H	820	484	336	2140	24	2116
RSyn0810M02M	410	242	168	866	12	854
RSyn0810M03M	615	363	252	1452	18	1434
RSyn0810M04M	820	484	336	2140	24	2116

Continued on next page

Continued from previous page

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
RSyn0815H	940	564	376	2430	44	2386
RSyn0815M02M	470	282	188	981	22	959
RSyn0815M03M	705	423	282	1647	33	1614
RSyn0815M04M	940	564	376	2430	44	2386
RSyn0820H	1020	604	416	2676	56	2620
RSyn0820M02M	510	302	208	1074	28	1046
RSyn0820M03M	765	453	312	1809	42	1767
RSyn0820M04M	1020	604	416	2676	56	2620
RSyn0830H	1240	744	496	3192	80	3112
RSyn0830M02M	620	372	248	1272	40	1232
RSyn0830M03M	930	558	372	2151	60	2091
RSyn0830M04M	1240	744	496	3192	80	3112
RSyn0830M	250	156	94	425	20	405
RSyn0840H	1440	864	576	3728	112	3616
RSyn0840M02M	720	432	288	1480	56	1424
RSyn0840M03M	1080	648	432	2508	84	2424
RSyn0840M04M	1440	864	576	3728	112	3616
RSyn0840M	280	176	104	484	28	456
SLay07H	476	392	84	609	0	609
SLay08H	632	520	112	812	0	812
SLay08M	184	72	112	252	0	252
SLay09H	810	666	144	1044	0	1044
SLay10H	1010	830	180	1305	0	1305
Syn20M03M	315	195	120	699	42	657

Continued on next page

Continued from previous page

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
Syn20M04M	420	260	160	1052	56	996
Syn30M02M	320	200	120	604	40	564
Syn30M03M	480	300	180	1041	60	981
Syn30M04M	640	400	240	1568	80	1488
Syn40M02M	420	260	160	812	56	756
Syn40M03M	630	390	240	1398	84	1314
Syn40M04M	840	520	320	2104	112	1992
Safety3	259	161	98	294	0	294
FLay06M	86	26	60	93	6	87
CLay0205H	260	210	50	365	40	325
CLay0205M	80	30	50	135	40	95
RSyn0815M03H	1347	1065	282	2217	33	2184
RSyn0815M04H	1796	1420	376	3190	44	3146
RSyn0820M03H	1467	1155	312	2448	42	2406
RSyn0820M04H	1956	1540	416	3528	56	3472
RSyn0820M	215	131	84	371	14	357
RSyn0830M03H	1758	1386	372	2934	60	2874
RSyn0830M04H	2344	1848	496	4236	80	4156
RSyn0840M03H	2040	1608	432	3447	84	3363
RSyn0840M04H	2720	2144	576	4980	112	4868
SLay07M	140	56	84	189	0	189
SLay09M	234	90	144	324	0	324
SLay10M	290	110	180	405	0	405
Syn15M04M	340	220	120	806	44	762

Continued on next page

Continued from previous page

Instance	Vars	C. Vars	Bin. Vars	Cons	Nl. Cons	L. Cons
Syn20M02M	210	130	80	406	28	378
Syn40M	130	90	40	226	28	198
RSyn0805M03H	1050	828	222	1698	9	1689
RSyn0810M03H	1185	933	252	1935	18	1917
stockcycle	480	48	432	97	0	97

Table A.4: Summary of Results (Solution Times) for Easy Instances

Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
alan	0.0	0.0	0.0	0.1	0.0
batch	0.1	0.1	0.5	0.5	0.1
batchdes	0.0	0.0	0.0	0.0	0.0
csched1	0.0	0.4	120.0	0.6	0.1
enpro48	3.5	5.1	2.8	3.3	0.8
enpro56	46.7	9.5	14.6	17.0	0.9
ex1223	0.0	0.0	0.2	0.3	0.0
ex1223a	0.0	0.0	0.0	0.0	0.0
ex1223b	0.0	0.0	0.2	0.2	0.0
ex3	0.0	0.0	0.0	0.0	0.0
ex4	0.2	0.9	120.5	1.8	0.3
fac1	0.0	0.0	0.1	0.2	0.0
fac2	0.1	0.2	-1	-1	0.1
fac3	0.1	0.2	1.7	1.9	0.1

Continued on next page

Continued from previous page

Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
gbd	0.0	0.0	0.0	0.0	0.0
m3	0.0	0.1	0.3	0.4	0.1
meanvarx	0.0	0.0	0.0	0.0	0.0
optprloc	0.1	0.8	120.4	1.5	0.2
prob02	0.0	0.0	0.0	0.0	0.0
prob03	0.0	0.0	0.0	0.0	0.0
ravem	0.5	0.6	4.3	3.8	0.2
ste14	0.0	0.0	0.2	0.2	0.0
ste38	0.0	0.0	0.0	0.0	0.0
stmiqp1	0.0	0.0	0.0	0.0	0.0
stmiqp2	0.0	0.0	-1	-1	0.0
stmiqp3	0.0	0.0	120.1	0.0	0.0
stmiqp4	0.0	0.0	120.1	0.0	0.0
stmiqp5	0.0	0.0	120.1	0.0	0.0
sttest1	0.0	0.0	120.0	0.1	0.0
sttest2	0.0	0.0	120.1	0.0	0.0
sttest3	0.0	0.0	120.1	0.0	0.0
sttest4	0.0	0.0	-1	-1	0.0
sttest5	0.0	0.0	120.2	0.0	0.0
sttest6	0.0	0.1	0.0	0.0	0.0
sttest8	0.0	0.0	120.1	0.0	0.0
sttestgr1	0.0	1.1	120.2	2.7	0.6
sttestgr3	0.0	1.0	120.1	1.4	0.2
sttestph4	0.0	0.0	120.1	0.0	0.0

Continued on next page

Continued from previous page

Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
synthes1-10	0.0	0.0	0.1	0.1	0.0
synthes1	0.0	0.0	0.1	0.1	0.0
synthes2	0.0	0.0	0.1	0.1	0.0
synthes3	0.0	0.1	0.4	0.3	0.0
tls2	0.5	3.5	4.1	4.1	0.2
trimloss2	0.2	1.0	3.8	3.1	0.2
BNS121612	3.9	13.2	6.3	5.1	0.8
BNS162010	11.8	17.8	7.5	5.8	3.2
BNS8125	0.8	0.7	2.6	2.3	0.2
BSBM8664	8.2	9.2	77.8	61.6	2.3
BSCH8664	38.9	23.9	115.9	90.0	5.3
Process20M	1.8	6.4	120.2	0.3	0.5
Process8CH	0.0	0.0	0.0	0.0	0.0
Process8C	0.0	0.1	0.0	-1	0.0
SLBM42	5.9	1.3	1.0	0.9	0.8
CLay0203H	2.6	4.5	40.7	33.0	1.2
CLay0203M	0.6	1.8	4.3	4.0	0.4
CLay0204H	50.4	39.9	52.4	40.3	5.1
CLay0204M	6.6	10.1	4.0	3.6	1.1
CLay0303H	3.9	12.7	53.0	32.6	4.4
CLay0303M	1.8	12.7	139.2	2.2	0.7
FLay02H	0.0	0.0	0.4	0.3	0.0
FLay02M	0.0	0.0	0.1	0.1	0.0
FLay03H	1.1	0.6	24.4	19.0	0.4

Continued on next page

Continued from previous page

Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
FLay03M	0.1	0.2	2.5	2.2	0.2
FLay04M	2.2	12.7	48.6	43.3	4.9
SLay04H	1.6	1.4	47.6	36.4	0.3
SLay04M	0.2	0.6	1.1	1.0	0.1
SLay05H	11.8	20.4	124.2	123.5	1.1
SLay05M	3.9	1.5	6.4	5.7	0.2
Syn05H	0.0	0.0	120.1	0.0	0.0
Syn05M02H	0.1	0.1	120.2	0.0	0.1
Syn05M02M	0.2	0.1	120.2	0.1	0.1
Syn05M03H	0.1	0.3	120.2	0.0	0.2
Syn05M03M	0.3	0.2	120.2	0.1	0.2
Syn05M04H	0.2	0.5	120.3	0.0	0.4
Syn05M04M	1.0	0.4	120.2	0.2	0.3
Syn05M	0.0	0.0	120.1	0.0	0.0
Syn10H	0.0	0.0	120.3	0.0	0.0
Syn10M02H	1.5	0.5	120.3	0.1	0.3
Syn10M02M	43.3	2.8	120.3	0.6	1.0
Syn10M03H	1.6	0.8	120.2	0.1	0.5
Syn10M04H	2.4	1.6	120.4	0.2	1.2
Syn10M	0.1	0.2	120.1	0.1	0.1
Syn15H	0.1	0.1	120.2	0.1	0.1
Syn15M02H	0.7	0.9	120.4	0.1	0.8
Syn15M03H	4.4	1.9	120.5	0.2	1.5
Syn15M04H	3.4	3.4	120.3	0.3	2.8

Continued on next page

Continued from previous page					
Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
Syn15M	0.3	0.3	120.1	0.5	0.2
Syn20H	0.2	0.2	120.2	0.1	0.1
Syn20M02H	20.6	1.4	120.5	0.4	1.2
Syn20M03H	25.2	3.1	120.4	0.5	2.1
Syn20M	2.0	3.1	120.2	1.4	1.0
Syn30H	1.2	0.7	120.3	0.4	0.5
Syn30M02H	59.9	4.0	120.8	1.0	2.2
Syn40H	5.2	1.5	120.5	0.7	0.8
Water0202R	1.7	1.5	2.4	1.2	0.9
Water0303R	59.7	154.0	160.4	149.5	43.9
Syn40M04H	1550.1	54.6	132.4	13.2	8.0

Table A.5: Summary of Results (Solution Times) for Moderate Instances

Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
m6	394.3	640.5	154.0	159.0	36.6
BatchStorageBM9964	60.7	333.4	146.7	140.6	46.0
BatchStorageCH9964	474.0	458.8	168.0	155.5	78.7
BatchStorage10BM10	1316.5	7125.4	254.8	206.9	540.0
BatchStorageBM10	1730.4	6894.1	255.0	226.9	887.2
SafetynorotationBM	83.7	183.5	130.0	126.4	36.4
BatchS101006M	163.6	3209.6	161.6	149.8	15.2
BatchS121208M	1436.5	5785.4	192.9	176.1	36.7
Continued on next page					

Continued from previous page

Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
FLay05M	324.4	10084.3	4904.1	3015.1	823.7
CLay0304H	394.3	692.9	1490.4	318.1	30.2
CLay0304M	256.7	540.2	511.3	1226.4	20.6
CLay0305M	298.1	927.9	213.6	203.8	43.8
RSyn0810M	448.5	402.7	154.8	8.2	13.9
RSyn0815M	402.7	143.9	127.5	13.2	16.0
SLay06H	1962.8	2077.4	148.8	139.6	10.2
Syn15M03M	2148.3	73.7	121.2	7.3	4.2
Syn30M	226.4	563.6	120.7	0.6	6.3
SafetyLayout4Conv	276.2	46.3	108.6	71.7	5.8
SafetyLayoutBM52	921.0	8.0	7.6	6.8	1.7
FLay04H	77.1	45.8	155.1	152.2	13.7
RSyn0805M02H	1398.5	16.3	123.1	10.2	3.1
RSyn0805M04H	798.0	12.7	120.2	2.2	7.7
RSyn0810M02H	843.6	12.2	126.0	5.3	3.9
RSyn0815M02H	541.1	10.5	124.6	3.8	3.1
RSyn0820M02H	2446.4	38.0	124.3	8.0	4.6
RSyn0830M02H	2000.0	20.2	126.4	6.5	5.6
RSyn0840M02H	1122.2	18.5	120.7	4.9	10.7
SLay06M	84.3	39.4	23.5	20.9	1.0
Syn10M03M	261.8	21.3	120.6	2.7	3.2
Syn10M04M	2089.7	59.5	121.5	2.0	7.1
Syn15M02M	394.1	20.4	121.0	0.4	1.2
Syn20M04H	62.0	6.1	120.9	0.6	3.3

Continued on next page

Continued from previous page					
Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
Syn30M03H	130.2	10.4	120.3	2.4	3.7
Syn30M04H	360.9	23.8	124.9	2.0	9.0
Syn40M02H	125.8	4.5	122.1	1.4	2.1
Syn40M03H	615.5	32.9	127.8	12.9	9.5
RSyn0810M04H	1248.0	33.9	123.3	5.1	15.1

Table A.6: Summary of Results (Solution Values) for Hard Instances

Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
csched2	-166102.0*	-158733.8	-165812.0	-165812.0	-164027.6
fo7	45.6	-1	26.0	24.5	20.7*
fo72	40.6	-1	17.7*	17.7*	17.7*
fo8	36.1	-1	-1	43.6	22.4*
fo9	-1	-1	-1	-1	23.5*
m7	220.5	-1	106.8*	106.8*	106.8*
o7	164.0	-1	-1	-1	131.7*
o72	132.7	-1	152.2	150.2	116.9*
tls12	-1	-1	-1	-1	-1
tls4	-1	-1	8.3*	8.3*	10.3
tls5	-1	-1	12.5	12.5	12.0
tls6	-1	-1	-1	-1	-1
tls7	-1	-1	-1	-1	-1
trimloss12	-1	-1	-1	-1	-1

Continued on next page

Continued from previous page

Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
trimloss4	11.3	-1	8.3*	8.3*	8.3*
trimloss5	12.6	-1	12.2	12.5	11.2
trimloss6	-1	-1	-1	-1	18.9
trimloss7	-1	-1	-1	-1	-1
safetyrotationCH	28380.7*	28380.7*	28380.7*	28380.7*	28380.7*
BatchS151208M	1543472.4	1543472.4	1543470.0*	1543470.0*	1543472.4
BatchS201210M	2295348.8*	-1	2295350.0	2295350.0	2295348.8*
CLay0305H	8092.5*	8497.5	8092.5*	8092.5*	8092.5*
FLay05H	64.5*	64.5*	64.5*	64.5*	64.5*
FLay06H	66.9	66.9	66.9	66.9	66.9
RSyn0805H	-6573.3	-1	-7174.2*	-7174.2*	-7174.2*
RSyn0805M02M	-2168.0	-1	-2238.4*	-2238.4*	-2238.4*
RSyn0805M03M	-2607.9	-1	-3068.8*	-3068.9*	-3068.9*
RSyn0805M04M	-6573.3	-1	-7174.2*	-7174.2*	-7174.2*
RSyn0810H	-1	-1	-6581.9*	-6581.9*	-6581.9*
RSyn0810M02M	-1129.3	-1	-1729.1	-1741.4*	-1741.4*
RSyn0810M03M	-2196.2	-1	-2722.5*	-2720.8	-2722.5*
RSyn0810M04M	-1	-1	-6581.9*	-6581.9*	-6581.9*
RSyn0815H	-1	-1	-1942.4	-1942.4	-3353.4
RSyn0815M02M	-493.7	-1	-1774.4*	-1774.4*	-1774.4*
RSyn0815M03M	-2284.7	-1	-2719.6	-2697.6	-2827.6
RSyn0815M04M	-1	-1	-1942.4	-1942.4	-3353.4
RSyn0820H	-1	-1	-1565.2	-2345.12	-2411.5
RSyn0820M02M	-154.7	-1	-1052.0	-1092.1*	-1026.9

Continued on next page

Continued from previous page

Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
RSyn0820M03M	-1014.6	-1	-1951.5	-1951.5	-1988.9
RSyn0820M04M	-1	-1	-1565.2	-2345.1	-2433.8
RSyn0830H	-1	-1	-247.1	-247.1	-2385.9
RSyn0830M02M	-121.3	-1	-730.5*	-730.5*	-716.5
RSyn0830M03M	-1249.3	-1	-1496.1	-696.9	-1469.3
RSyn0830M04M	-1	-1	-247.1	-247.1	-2385.9
RSyn0830M	-402.4	-1	-510.1*	-510.1*	-506.7
RSyn0840H	-1	-1	-2347.7	-2347.7	-2448.7
RSyn0840M02M	-58.7	-1	-718.4	-735.0	-696.0
RSyn0840M03M	-1	-1	-2742.7*	-2742.7*	-2674.1
RSyn0840M04M	-1	-1	-2347.7	-2347.7	-2448.7
RSyn0840M	-52.9	-1	-325.6*	-325.6*	-289.1
SLay07H	64748.8*	64748.8*	64748.8*	64748.8*	64748.8*
SLay08H	86744.6	90110.7	84960.2*	84960.2*	84960.2*
SLay08M	312377.7	85115.0	84960.2*	84960.2*	84960.2*
SLay09H	135952.1	116955.3	107806.0	107806.0	107805.8*
SLay10H	194758.1	-1	129580.0*	129580.0*	138730.4
Syn20M03M	-2532.9	-1394.6	-2647.0*	-2647.0*	-2647.0*
Syn20M04M	-1695.1	-1056.3	-3532.7*	-3532.7*	-3532.7*
Syn30M02M	-352.4	-13.1	-399.7*	-399.7*	-399.7*
Syn30M03M	-610.1	-17.1	-654.2	-654.2	-648.7
Syn30M04M	-798.2	-31.1	-865.7	-865.7	-852.2
Syn40M02M	-321.7	-13.1	-388.8	-388.8	-374.5
Syn40M03M	-270.3	-17.1	-395.1	-395.1	-358.9

Continued on next page

Continued from previous page					
Instance	MINLP-BB	vanilla	BONMIN	BONMIN-v2	FilMINT
Syn40M04M	-837.5	-31.1	-901.8	-901.8	-795.7
Safety3	195782.3	38435.0	27803.0*	27803.0*	27803.0*
FLay06M	66.9*	66.9*	66.9*	66.9*	66.9*
CLay0205H	8092.5*	8092.5*	8092.5*	8092.5*	8092.5*
CLay0205M	25611.3	8092.5*	8092.5*	8092.5*	8092.5*
RSyn0815M03H	-2807.9	-2827.9*	-2827.9*	-2827.9*	-2827.9*
RSyn0815M04H	-3375.9	-3398.7	-3410.9	-3410.9	-3405.8
RSyn0820M03H	-2028.8*	-2028.8*	-2028.8*	-2028.8*	-2022.6
RSyn0820M04H	-1	-2450.8	-2450.8	-2450.8	99.4
RSyn0820M	-1022.0	-1150.3	-1150.3	-1150.3	-1146.3
RSyn0830M03H	-1543.1	-1543.1	-1543.1	-1543.1	229.3
RSyn0830M04H	-1	-2527.3	-2529.1	-2529.1	102.9
RSyn0840M03H	-2742.6	-2742.6	-2742.7	-2742.7	-1031.9
RSyn0840M04H	-1	-2564.5	-2564.5	-2564.5	143.3
SLay07M	77483.4	64748.8*	64748.8*	64748.8*	64748.8*
SLay09M	438359.2	114444.9	107806.0	107806.0	107805.8*
SLay10M	682209.3	-1	129580.0	129580.0	129579.9*
Syn15M04M	-4909.5	-4937.5*	-4937.5*	-4937.5*	-4937.5*
Syn20M02M	-1703.0	-1752.1*	-1752.1*	-1752.1*	-1752.1*
Syn40M	-39.7	-63.9	-67.7*	-67.7*	-67.7*
RSyn0805M03H	-3022.1	-3068.9*	-3068.9*	-3068.9*	-3068.9*
RSyn0810M03H	-2722.4	-2722.4	-2722.5	-2722.5	-2722.4
stockcycle	119948.7*	206468.2	119949.0	119949.0	159828.7

For Tables A.7–A.9, the row-headings are described as follows:

- #I-Sol: Number of integer solution found.
- #F-Sol: Number of feasible solution found.
- #IPCut: Number of MILP-based cuts added.
- #Nodes: Number of nodes evaluated.
- #LPs: Number of LPs solved.
- #FinalC: Number of constraints at the end of run.
- #OrigC: Number of original linear constraints in the instance.

Table A.7: Statistics of runs with FilMINT for Easy Instances

Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
alan	5	5	4	9	15	18	7
batch	3	2	2	35	55	96	72
batchdes	3	2	0	7	11	23	18
csched1	16	16	0	125	142	34	22
enpro48	4	4	0	236	273	259	213
enpro56	4	2	0	285	304	215	190
ex1223	3	3	0	9	16	24	9
ex1223a	3	3	0	3	4	11	5
ex1223b	3	3	0	11	18	21	5
ex3	1	1	0	1	2	37	26
ex4	4	4	0	133	148	236	5

Continued on next page

Continued from previous page

Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
fac1	2	2	10	5	8	32	18
fac2	10	10	7	29	41	51	33
fac3	6	6	8	23	33	49	33
gbd	2	2	0	1	2	5	4
m3	3	3	0	68	98	140	37
meanvarx	4	4	0	9	14	44	44
optprloc	3	2	0	127	141	227	5
prob02	2	1	0	1	2	14	3
prob03	3	2	0	3	4	3	0
ravem	4	2	0	51	67	206	185
ste14	3	3	0	9	16	24	9
ste38	2	2	0	1	2	6	2
stmiqp1	3	3	0	9	13	6	1
stmiqp2	4	4	4	7	21	21	3
stmiqp3	4	4	0	2	5	5	1
stmiqp4	1	1	0	1	2	6	4
stmiqp5	2	2	0	1	2	15	13
sttest1	7	7	0	15	23	10	1
sttest2	3	3	0	3	6	6	2
sttest3	2	2	1	3	10	15	10
sttest4	2	2	0	4	5	7	5
sttest5	1	1	12	9	17	30	11
sttest6	1	1	10	11	24	26	5
sttest8	4	4	0	3	7	25	20

Continued on next page

Continued from previous page

Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
sttestgr1	64	64	0	1691	1761	43	5
sttestgr3	5	5	0	413	421	29	20
sttestph4	2	2	0	1	3	10	10
synthes1-10	5	5	0	5	18	23	4
synthes1	5	5	0	5	18	23	4
synthes2	4	4	0	11	21	25	11
synthes3	4	4	1	19	31	37	19
tls2	7	2	12	184	208	71	22
trimloss2	7	1	10	175	202	76	22
BNS121612	5	4	172	123	144	563	416
BNS162010	5	4	100	682	759	745	680
BNS8125	8	5	0	80	94	210	216
BSBM8664	20	18	0	329	376	431	370
BSCH8664	21	20	0	384	452	671	598
Process20M	12	12	6	395	453	206	87
Process8CH	2	2	0	7	13	81	105
Process8C	2	2	0	1	4	78	105
SLBM42	4	2	13	293	335	385	52
CLay0203H	10	5	0	277	329	439	108
CLay0203M	8	5	8	209	271	302	30
CLay0204H	5	5	0	1308	1475	603	202
CLay0204M	6	5	22	1040	1142	344	58
CLay0303H	13	3	0	925	1151	584	114
CLay0303M	20	5	0	521	602	322	30

Continued on next page

Continued from previous page

Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
FLay02H	2	2	0	7	12	62	49
FLay02M	3	3	0	7	14	23	9
FLay03H	9	9	0	105	140	239	141
FLay03M	11	11	3	101	144	135	21
FLay04M	143	143	22	2513	3139	361	38
SLay04H	9	9	0	139	149	185	174
SLay04M	8	8	13	119	133	77	54
SLay05H	11	11	0	359	372	302	290
SLay05M	9	9	9	145	159	110	90
Syn05H	3	3	0	1	3	38	55
Syn05M02H	2	2	0	3	7	111	145
Syn05M02M	6	6	2	19	34	125	95
Syn05M03H	2	2	0	3	7	189	240
Syn05M03M	6	6	4	37	57	252	165
Syn05M04H	2	2	1	3	7	282	350
Syn05M04M	8	8	4	43	68	371	250
Syn05M	2	2	2	11	23	42	25
Syn10H	3	3	0	3	4	71	106
Syn10M02H	3	3	0	5	11	235	282
Syn10M02M	5	5	8	367	436	380	186
Syn10M03H	3	3	0	3	6	374	468
Syn10M04H	3	3	0	9	17	609	684
Syn10M	4	4	1	39	49	64	48
Syn15H	3	3	0	3	7	119	170

Continued on next page

Continued from previous page							
Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
Syn15M02H	3	3	0	3	6	339	445
Syn15M03H	2	2	0	5	9	578	735
Syn15M04H	3	3	0	3	6	856	1070
Syn15M	5	5	2	87	117	161	78
Syn20H	3	3	0	5	13	185	219
Syn20M02H	4	4	0	7	15	452	578
Syn20M03H	2	2	2	9	16	776	957
Syn20M	9	9	3	601	682	272	99
Syn30H	3	3	0	5	22	305	325
Syn30M02H	6	6	0	5	17	741	860
Syn40H	6	6	0	15	51	435	438
Water0202R	5	5	0	29	35	290	283
Water0303R	29	29	0	513	556	587	556
Syn40M04H	6	6	15	143	159	2456	2792

Table A.8: Statistics of runs with FilMINT for Moderate Instances

Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
m6	4	4	0	10255	11319	360	145
BSBM9964	17	16	0	3477	3809	733	646
BSCH9964	29	28	0	4296	4641	1136	1047
BS10BM1064	99	98	0	35013	37859	963	798
BSBM1064	117	115	0	52257	57030	989	798
Continued on next page							

Continued from previous page

Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
SafetynoBM	29	21	30	21361	22693	228	156
BS101006M	13	11	0	1648	1690	807	1018
BS121208M	11	8	0	2408	2453	1200	1510
FLay05M	6540	6540	200	78023	97978	1820	60
CLay0304H	31	5	0	6412	7418	681	210
CLay0304M	59	6	0	7057	8517	719	58
CLay0305M	35	17	29	16498	18654	788	95
RS0810M	7	7	6	9552	9614	320	306
RS0815M	6	6	4	8980	9120	454	336
SLay06H	19	19	0	1843	1884	452	435
Syn15M03M	12	12	7	1471	1547	706	504
Syn30M	10	10	10	3248	3600	451	147
SL4Convex	4	3	0	785	941	749	196
SLBM52	7	5	5	1127	1219	298	85
FLay04H	84	84	0	2455	2923	542	278
RS0805M02H	5	5	0	93	104	984	1039
RS0805M04H	4	4	5	28	38	2302	2426
RS0810M02H	5	5	0	112	125	1122	1176
RS0815M02H	4	4	0	73	83	1226	1339
RS0820M02H	6	6	0	91	104	1329	1472
RS0830M02H	2	2	0	1	5	1541	1754
RS0840M02H	9	9	0	50	67	1806	2050
SLay06M	14	14	26	452	482	175	135
Syn10M03M	9	9	14	808	911	732	324

Continued on next page

Continued from previous page							
Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
Syn10M04M	7	7	18	1773	1928	865	492
Syn15M02M	7	7	4	467	501	390	291
Syn20M04H	2	2	2	1	4	1122	1396
Syn30M03H	3	3	4	1	6	1179	1425
Syn30M04H	4	4	9	19	34	1927	2080
Syn40M02H	7	7	0	27	47	1055	1156
Syn40M03H	16	16	8	55	87	1856	1914
RS0810M04H	11	11	9	61	76	2656	2760

Table A.9: Statistics of runs with FilMINT for Difficult Instances

Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
csched2	787	757	0	86939	99164	409	137
fo7	8	4	0	361871	402829	675	197
fo72	2	2	0	135531	150574	705	197
fo8	3	3	0	386257	425219	871	257
fo9	11	11	0	1000000	1084099	685	325
m7	5	5	0	32801	36263	469	197
o7	8	8	0	1000000	1096725	462	197
o72	21	20	0	1000000	1104910	892	197
tls12	12	0	839	7393	15238	1747	372
tls4	12	1	161	779	1084	265	60
tls5	164	1	1321	92645	189313	886	85
Continued on next page							

Continued from previous page

Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
tls6	166	0	2311	141517	262047	1007	114
tls7	76	0	7084	256446	395486	980	147
trimloss12	48	0	763	19321	28732	1164	372
trimloss4	104	3	383	77668	96957	473	60
trimloss5	266	3	1608	209326	373133	869	85
trimloss6	343	3	2542	150870	280608	1008	114
trimloss7	69	0	5094	170604	264523	998	147
safetynoCH	33	26	0	163930	168541	533	456
BS151208M	14	11	0	3589	3653	1407	1780
BS201210M	12	11	0	8457	8606	1843	2326
CLay0305H	24	18	0	30616	32310	1107	335
FLay05H	5499	5499	0	100309	121112	1990	460
FLay06H	26717	26717	0	533014	622308	2174	687
RS0805H	16	16	17	60696	62364	2119	1874
RS0805M02M	35	35	10	28588	29030	902	763
RS0805M03M	12	12	16	49240	50845	1450	1275
RS0805M04M	16	16	17	60696	62364	2119	1874
RS0810H	28	28	35	719793	741129	2654	2116
RS0810M02M	61	61	16	505104	511097	1122	854
RS0810M03M	34	34	22	668174	696373	1953	1434
RS0810M04M	28	28	35	719793	741129	2654	2116
RS0815H	15	15	14	1000000	1018787	2612	2386
RS0815M02M	56	56	12	1000000	1017296	1054	959
RS0815M03M	83	83	14	1000000	1015334	1795	1614

Continued on next page

Continued from previous page

Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
RS0815M04M	15	15	14	1000000	1018787	2612	2386
RS0820H	23	23	27	767435	789570	2962	2620
RS0820M02M	42	42	13	1000000	1016373	1257	1046
RS0820M03M	46	46	17	1000000	1025138	1942	1767
RS0820M04M	30	30	27	1000000	1027336	2962	2620
RS0830H	112	112	55	766117	832950	3905	3112
RS0830M02M	73	73	26	1000000	1025981	1632	1232
RS0830M03M	108	108	62	1000000	1030812	2683	2091
RS0830M04M	111	111	55	493201	545698	3886	3112
RS0830M	287583	287583	19	390180	655843	624	405
RS0840H	93	93	48	583004	612951	4569	3616
RS0840M02M	56	56	19	1000000	1029620	1800	1424
RS0840M03M	60	60	43	1000000	1020045	2685	2424
RS0840M04M	93	93	48	580771	610625	4569	3616
RS0840M	143249	143249	21	192000	325028	640	456
SLay07H	37	37	0	6009	6184	634	609
SLay08H	125	125	0	35100	36287	893	812
SLay08M	36	36	59	2355	2512	299	252
SLay09H	236	236	0	132012	136247	1226	1044
SLay10H	288	288	0	615869	626612	1522	1305
Syn20M03M	20	20	11	110949	114777	1222	657
Syn20M04M	43	43	14	583549	600367	1503	996
Syn30M02M	38	38	20	298873	319680	1364	564
Syn30M03M	28	28	41	1000000	1035901	1481	981

Continued on next page

Continued from previous page

Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
Syn30M04M	30	30	52	1000000	1049147	2195	1488
Syn40M03M	43	43	38	1000000	1026985	1646	1314
Syn40M04M	16	16	30	1000000	1040641	2561	1992
Safety3	710	710	41	1000000	1089911	802	294
FLay06M	76932	76932	3232	1000000	1246999	1660	87
CLay0205H	11	8	0	17647	18798	601	325
CLay0205M	16	12	0	11210	11959	547	95
RS0815M03H	4	4	8	159	170	2019	2184
RS0815M04H	9	9	12	113	133	2941	3146
RS0820M03H	79	79	9	462	486	2209	2406
RS0820M04H	2	2	14	1	6	3124	3472
RS0820M	562489	562489	4	613869	1127148	542	357
RS0830M03H	2	2	9	1	5	2538	2874
RS0830M04H	2	2	19	1	6	3721	4156
RS0840M03H	2	2	11	1	6	2942	3363
RS0840M04H	2	2	12	1	5	4285	4868
SLay07M	24	24	25	889	949	226	189
SLay09M	241	241	147	42508	46520	491	324
SLay10M	1044	1044	254	977341	1037017	908	405
Syn15M04M	23	23	9	4395	4647	1107	762
Syn20M02M	12	12	8	14721	15150	727	378
Syn40M	12	12	8	43957	45992	493	198
RS0805M03H	6	6	1	68	83	1617	1689
RS0810M03H	8	8	8	159	176	1838	1917

Continued on next page

Continued from previous page							
Instance	#I-Sol	#F-Sol	#IPCut	#Nodes	#LPs	#FinalC	#OrigC
stockcycle	9785	9785	0	1000000	1057031	1104	97

For Tables A.10–A.15, the row-headings are described as follows:

- Vars: Number of variables.
- C.Vars: Number of continuous variables.
- #NLPs: Number of NLPs (of a particular kind) solved.
- A.Time: Average time to solve an NLP.
- Cuts: Number of linearizations from the solution of NLP added.

Table A.10: Statistics related to (NLP(y^k)) and (NLPR(l, u)) for Easy Instances

			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
alan	8	4	5	0.0	6	1	0.0
batch	46	22	3	0.0	7	1	0.0
batchdes	19	10	3	0.0	6	1	0.0
csched1	76	13	16	0.0012	17	1	0.0
enpro48	153	61	4	0.0114	10	1	0.02
enpro56	127	54	4	0.0088	8	1	0.01
ex1223	11	7	3	0.0	8	1	0.0

Continued on next page

Continued from previous page

			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
ex1223a	7	3	3	0.0	6	1	0.0
ex1223b	7	3	3	0.0	9	1	0.0
ex3	32	24	1	0.0	10	1	0.0
ex4	36	11	4	0.0029	40	1	0.0
fac1	22	16	2	0.0	3	1	0.0
fac2	66	54	10	0.0015	7	1	0.0
fac3	66	54	6	0.0	6	1	0.01
gbd	4	1	2	0.0	2	1	0.0
m3	26	20	3	0.0	16	1	0.0
meanvarx	35	21	4	0.0	5	1	0.0
optprloc	30	5	3	0.002	37	1	0.0
prob02	6	0	2	0.0	10	1	0.0
prob03	2	0	3	0.0	2	1	0.0
ravem	112	58	4	0.0033	8	1	0.0
ste14	11	7	3	0.0	8	1	0.0
ste38	4	2	2	0.0	4	1	0.0
stmiqp1	5	0	3	0.0	4	1	0.0
stmiqp2	4	0	4	0.0	4	1	0.0
stmiqp3	2	0	4	0.0	3	1	0.0
stmiqp4	6	3	1	0.0	2	1	0.0
stmiqp5	7	5	2	0.0	2	1	0.0
sttest1	5	0	7	0.0	8	1	0.0
sttest2	6	0	3	0.0	3	1	0.0

Continued on next page

Continued from previous page

			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
sttest3	13	0	2	0.0	3	1	0.0
sttest4	6	0	2	0.0	2	1	0.0
sttest5	10	0	1	0.0	2	1	0.0
sttest6	10	0	1	0.0	2	1	0.0
sttest8	24	0	4	0.0	4	1	0.0
sttestgr1	10	0	64	0.0002	65	1	0.0
sttestgr3	20	0	5	0.0	6	1	0.0
sttestph4	3	0	2	0.0	3	1	0.0
synthes1-10	6	3	5	0.0	8	1	0.0
synthes1	6	3	5	0.0	8	1	0.0
synthes2	11	6	4	0.0	8	1	0.0
synthes3	17	9	4	0.0	10	1	0.0
tls2	37	4	7	0.0	7	1	0.0
trimloss2	37	6	7	0.0	8	1	0.0
BNS121612	248	56	5	0.0033	11	1	0.02
BNS162010	272	72	5	0.0117	11	1	0.03
BNS8125	100	40	8	0.0044	13	1	0.01
BSBM8664	132	79	20	0.0055	38	1	0.01
BSCH8664	318	265	21	0.0333	41	1	0.08
Process20M	65	45	12	0.0	22	1	0.01
Process8CH	71	63	2	0.0033	6	1	0.0
Process8C	71	63	2	0.0	5	1	0.01
SLBM42	70	38	4	0.0025	55	1	0.01

Continued on next page

Continued from previous page

			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
CLay0203H	90	72	10	0.0054	59	1	0.0
CLay0203M	30	12	8	0.001	51	1	0.0
CLay0204H	164	132	5	0.0092	54	1	0.02
CLay0204M	52	20	6	0.001	44	1	0.0
CLay0303H	99	78	13	0.0046	77	1	0.01
CLay0303M	33	12	20	0.0004	106	1	0.0
FLay02H	46	42	2	0.0	6	1	0.0
FLay02M	14	10	3	0.0	6	1	0.0
FLay03H	122	110	9	0.0082	21	1	0.0
FLay03M	26	14	11	0.0008	21	1	0.0
FLay04M	42	18	143	0.0006	167	1	0.0
SLay04H	140	116	9	0.004	10	1	0.0
SLay04M	44	20	8	0.0	9	1	0.0
SLay05H	230	190	11	0.0083	12	1	0.01
SLay05M	70	30	9	0.002	10	1	0.0
Syn05H	42	37	3	0.0	4	1	0.0
Syn05M02H	104	84	2	0.0033	7	1	0.06
Syn05M02M	60	40	6	0.0014	8	1	0.01
Syn05M03H	156	126	2	0.01	10	1	0.11
Syn05M03M	90	60	6	0.0025	15	1	0.01
Syn05M04H	208	168	2	0.0233	13	1	0.26
Syn05M04M	120	80	8	0.0042	17	1	0.02
Syn05M	20	15	2	0.0	3	1	0.0

Continued on next page

Continued from previous page							
			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
Syn10H	77	67	3	0.0	6	1	0.0
Syn10M02H	194	154	3	0.015	22	1	0.15
Syn10M02M	110	70	5	0.0017	19	1	0.01
Syn10M03H	291	231	3	0.0375	18	1	0.31
Syn10M04H	388	308	3	0.0875	39	1	0.72
Syn10M	35	25	4	0.0	7	1	0.0
Syn15H	121	106	3	0.0075	12	1	0.01
Syn15M02H	302	242	3	0.0425	23	1	0.55
Syn15M03H	453	363	2	0.1033	34	1	1.09
Syn15M04H	604	484	3	0.1875	45	1	1.93
Syn15M	55	40	5	0.0029	16	1	0.0
Syn20H	151	131	3	0.008	24	1	0.03
Syn20M02H	382	302	4	0.066	31	1	0.73
Syn20M03H	573	453	2	0.1533	45	1	1.48
Syn20M	65	45	9	0.0027	20	1	0.01
Syn30H	228	198	3	0.0317	29	1	0.23
Syn30M02H	576	456	6	0.1238	71	1	1.07
Syn40H	302	262	6	0.036	35	1	0.26
Water0202R	195	188	5	0.0883	6	1	0.09
Water0303R	384	370	29	0.6487	30	1	0.79
Syn40M04H	1528	1208	6	0.2278	125	1	2.29

Table A.11: Statistics related to Fixfrac and ECP Methods for Easy Instances

			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
alan	8	4	4	0.0	0	1	0.0	1
batch	46	22	16	0.0	14	3	0.0	5
batchdes	19	10	3	0.0	0	1	0.0	1
csched1	76	13	6	0.0	0	1	0.0012	1
enpro48	153	61	30	0.0003	30	3	0.0114	6
enpro56	127	54	12	0.0	11	4	0.0088	6
ex1223	11	7	4	0.0	3	2	0.0	4
ex1223a	7	3	1	0.0	0	1	0.0	0
ex1223b	7	3	6	0.0	2	3	0.0	5
ex3	32	24	0	0.0	0	0	0.0	0
ex4	36	11	8	0.0	135	3	0.0029	57
fac1	22	16	2	0.0	0	1	0.0	1
fac2	66	54	13	0.0	0	3	0.0015	3
fac3	66	54	12	0.0	0	2	0.0	2
gbd	4	1	0	0.0	0	0	0.0	0
m3	26	20	38	0.0	84	2	0.0	2
meanvarx	35	21	4	0.0	0	1	0.0	1
optprloc	30	5	9	0.0	147	2	0.002	39
prob02	6	0	0	0.0	0	0	0.0	0
prob03	2	0	1	0.0	0	1	0.0	0
ravem	112	58	11	0.0	10	2	0.0033	3
ste14	11	7	4	0.0	3	2	0.0	4

Continued on next page

Continued from previous page

			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
ste38	4	2	0	0.0	0	0	0.0	0
stmiqp1	5	0	4	0.0	0	1	0.0	1
stmiqp2	4	0	12	0.0	0	10	0.0	10
stmiqp3	2	0	1	0.0	0	1	0.0	1
stmiqp4	6	3	0	0.0	0	0	0.0	0
stmiqp5	7	5	0	0.0	0	0	0.0	0
sttest1	5	0	7	0.0	0	1	0.0	1
sttest2	6	0	2	0.0	0	2	0.0	1
sttest3	13	0	5	0.0	0	5	0.0	5
sttest4	6	0	2	0.0	0	1	0.0	0
sttest5	10	0	9	0.0	0	5	0.0	5
sttest6	10	0	14	0.0	0	9	0.0	9
sttest8	24	0	1	0.0	0	1	0.0	1
sttestgr1	10	0	32	0.0	0	1	0.0002	1
sttestgr3	20	0	19	0.0	0	3	0.0	3
sttestph4	3	0	0	0.0	0	0	0.0	0
synthes1-10	6	3	5	0.0	5	4	0.0	6
synthes1	6	3	5	0.0	5	4	0.0	6
synthes2	11	6	10	0.0	4	5	0.0	2
synthes3	17	9	16	0.0	4	6	0.0	3
tls2	37	4	12	0.0	23	3	0.0	6
trimloss2	37	6	15	0.0	30	2	0.0	4
BNS121612	248	56	5	0.0	5	1	0.0033	2

Continued on next page

Continued from previous page

			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
BNS162010	272	72	6	0.0	6	1	0.0117	2
BNS8125	100	40	5	0.0	5	1	0.0044	2
BSBM8664	132	79	27	0.0	25	2	0.0055	3
BSCH8664	318	265	43	0.0002	42	3	0.0333	4
Process20M	65	45	52	0.0	92	6	0.0	13
Process8CH	71	63	3	0.0	4	1	0.0033	1
Process8C	71	63	2	0.0	0	2	0.0	3
SLBM42	70	38	39	0.0	256	4	0.0025	8
CLay0203H	90	72	43	0.0	262	3	0.0054	14
CLay0203M	30	12	40	0.0	189	13	0.001	23
CLay0204H	164	132	55	0.0002	406	8	0.0092	32
CLay0204M	52	20	45	0.0002	240	4	0.001	24
CLay0303H	99	78	75	0.0	458	15	0.0046	45
CLay0303M	33	12	37	0.0	231	4	0.0004	5
FLay02H	46	42	3	0.0	6	1	0.0	0
FLay02M	14	10	3	0.0	6	1	0.0	1
FLay03H	122	110	27	0.0	75	2	0.0082	1
FLay03M	26	14	29	0.0	87	1	0.0008	2
FLay04M	42	18	57	0.0	179	8	0.0006	15
SLay04H	140	116	10	0.0	0	1	0.004	1
SLay04M	44	20	9	0.0	0	1	0.0	1
SLay05H	230	190	5	0.0	0	1	0.0083	1
SLay05M	70	30	5	0.0	0	1	0.002	1

Continued on next page

Continued from previous page

			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
Syn05H	42	37	0	0.0	0	0	0.0	0
Syn05M02H	104	84	1	0.0	1	1	0.0033	1
Syn05M02M	60	40	9	0.0	14	1	0.0014	2
Syn05M03H	156	126	1	0.0	2	1	0.01	2
Syn05M03M	90	60	12	0.0	54	2	0.0025	11
Syn05M04H	208	168	1	0.0	3	1	0.0233	2
Syn05M04M	120	80	15	0.0	73	4	0.0042	23
Syn05M	20	15	8	0.0	8	3	0.0	2
Syn10H	77	67	1	0.0	0	1	0.0	0
Syn10M02H	194	154	2	0.0	13	1	0.015	1
Syn10M02M	110	70	60	0.0	182	7	0.0017	10
Syn10M03H	291	231	1	0.0	10	1	0.0375	3
Syn10M04H	388	308	4	0.0025	49	1	0.0875	4
Syn10M	35	25	18	0.0	6	4	0.0	0
Syn15H	121	106	1	0.0	1	1	0.0075	1
Syn15M02H	302	242	1	0.0	2	1	0.0425	0
Syn15M03H	453	363	2	0.0	6	1	0.1033	0
Syn15M04H	604	484	1	0.0	4	1	0.1875	0
Syn15M	55	40	33	0.0	57	2	0.0029	5
Syn20H	151	131	4	0.0	24	2	0.008	3
Syn20M02H	382	302	3	0.0	13	1	0.066	1
Syn20M03H	573	453	4	0.0	28	1	0.1533	1
Syn20M	65	45	69	0.0	175	6	0.0027	15

Continued on next page

Continued from previous page

			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
Syn30H	228	198	12	0.0	73	3	0.0317	5
Syn30M02H	576	456	4	0.0	53	2	0.1238	12
Syn40H	302	262	26	0.0	124	4	0.036	7
Water0202R	195	188	12	0.0025	0	1	0.0883	1
Water0303R	384	370	30	0.0193	0	1	0.6487	1
Syn40M04H	1528	1208	14	0.0	211	3	0.2278	0

Table A.12: Statistics related to (NLP(y^k)) and (NLPR(l, u)) for Moderate Instances

			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
m6	86	56	4	0.0044	32	1	0.0
BSBM9964	205	125	17	0.012	30	1	0.05
BSCH9964	527	447	29	0.0927	49	1	0.31
BS10BM1064	238	149	99	0.0162	170	1	0.04
BSBM1064	238	149	117	0.0162	214	1	0.06
SafetynoBM	168	108	29	0.0031	54	1	0.01
BS101006M	278	149	13	0.0193	26	1	0.04
BS121208M	406	203	11	0.0392	21	1	0.08
FLay05M	62	22	6540	0.0013	5879	1	0.0
CLay0304H	176	140	31	0.0161	159	1	0.02
CLay0304M	56	20	59	0.0013	233	1	0.0

Continued on next page

Continued from previous page

			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
CLay0305M	85	30	35	0.0037	226	1	0.0
RSyn0810M	185	111	7	0.0053	10	1	0.01
RSyn0815M	205	126	6	0.011	16	1	0.03
SLay06H	342	282	19	0.014	19	1	0.02
Syn15M03M	255	165	12	0.0126	48	1	0.06
Syn30M	100	70	10	0.0033	37	1	0.01
SL4Convex	182	150	4	0.0167	52	1	0.0
SLBM52	110	60	7	0.0043	86	1	0.0
FLay04H	234	210	84	0.0109	142	1	0.02
RS0805M02H	700	552	5	0.18	14	1	0.91
RS0805M04H	1400	1104	4	0.752	19	1	2.43
RS0810M02H	790	622	5	0.23	23	1	1.15
RS0815M02H	898	710	4	0.26	35	1	0.75
RS0820M02H	978	770	6	0.3071	38	1	1.02
RS0830M02H	1172	924	2	0.4133	40	1	4.08
RS0840M02H	1360	1072	9	0.621	78	1	3.21
SLay06M	102	42	14	0.002	15	1	0.01
Syn10M03M	165	105	9	0.0071	33	1	0.02
Syn10M04M	220	140	7	0.0114	46	1	0.03
Syn15M02M	170	110	7	0.005	30	1	0.03
Syn20M04H	764	604	2	0.21	56	1	2.58
Syn30M03H	864	684	3	0.23	74	1	2.62
Syn30M04H	1152	912	4	0.4414	105	1	5.15

Continued on next page

Continued from previous page

			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
Syn40M02H	764	604	7	0.091	65	1	0.58
Syn40M03H	1146	906	16	0.1929	108	1	4.8
RS0810M04H	1580	1244	11	0.7783	51	1	3.07

Table A.13: Statistics related to Fixfrac and ECP Methods for Moderate Instances

			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
m6	86	56	57	0.0	208	5	0.0044	13
BSBM9964	205	125	76	0.0	76	8	0.012	16
BSCH9964	527	447	45	0.0	45	4	0.0927	8
BS10BM1064	238	149	79	0.0001	79	9	0.0162	15
BSBM1064	238	149	81	0.0	81	6	0.0162	9
SafetynoBM	168	108	18	0.0	28	3	0.0031	4
BS101006M	278	149	5	0.0	5	1	0.0193	2
BS121208M	406	203	5	0.0	5	1	0.0392	2
FLay05M	62	22	84	0.0	359	7	0.0013	19
CLay0304H	176	140	88	0.0002	603	7	0.0161	15
CLay0304M	56	20	96	0.0001	716	5	0.0013	14
CLay0305M	85	30	60	0.0002	563	6	0.0037	43
RSyn0810M	185	111	57	0.0002	25	8	0.0053	2
RSyn0815M	205	126	70	0.0	128	4	0.011	4

Continued on next page

Continued from previous page								
			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
SLay06H	342	282	16	0.0	0	1	0.014	0
Syn15M03M	255	165	42	0.0	149	7	0.0126	13
Syn30M	100	70	74	0.0	293	5	0.0033	25
SL4Convex	182	150	65	0.0	632	5	0.0167	12
SLBM52	110	60	32	0.0	133	7	0.0043	38
FLay04H	234	210	55	0.0	173	7	0.0109	13
RS0805M02H	700	552	5	0.0	11	1	0.18	4
RS0805M04H	1400	1104	4	0.0	21	1	0.752	6
RS0810M02H	790	622	7	0.0014	42	1	0.23	5
RS0815M02H	898	710	5	0.0	22	1	0.26	2
RS0820M02H	978	770	6	0.0	28	1	0.3071	3
RS0830M02H	1172	924	2	0.0	33	1	0.4133	11
RS0840M02H	1360	1072	7	0.0	53	1	0.621	8
SLay06M	102	42	5	0.0	0	1	0.002	1
Syn10M03M	165	105	71	0.0	403	5	0.0071	14
Syn10M04M	220	140	72	0.0	361	7	0.0114	25
Syn15M02M	170	110	29	0.0	62	3	0.005	3
Syn20M04H	764	604	1	0.0	10	1	0.21	1
Syn30M03H	864	684	2	0.0	52	1	0.23	7
Syn30M04H	1152	912	9	0.0	218	3	0.4414	25
Syn40M02H	764	604	16	0.0	166	3	0.091	11
Syn40M03H	1146	906	26	0.0004	339	1	0.1929	0
RS0810M04H	1580	1244	5	0.0	76	1	0.7783	11

Table A.14: Statistics related to (NLP(y^k)) and (NLPR(l, u)) for Difficult Instances

			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
csched2	400	92	787	0.0585	687	1	0.44
fo7	114	72	8	0.0094	80	1	0.01
fo72	114	72	2	0.0067	32	1	0.02
fo8	146	90	3	0.0136	32	1	0.02
fo9	182	110	11	0.012	64	1	0.03
m7	114	72	5	0.005	46	1	0.0
o7	114	72	8	0.0069	55	1	0.01
o72	114	72	21	0.0054	102	1	0.01
tls12	812	144	12	0.015	151	1	0.33
tls4	105	16	12	0.0	29	1	0.01
tls5	161	25	164	0.0005	392	1	0.03
tls6	215	36	166	0.0016	592	1	0.04
tls7	345	49	76	0.0024	376	1	0.12
trimloss12	812	156	48	0.0284	547	1	0.47
trimloss4	105	20	104	0.0004	228	1	0.01
trimloss5	161	30	266	0.0016	624	1	0.03
trimloss6	215	42	343	0.0017	1064	1	0.04
trimloss7	345	56	69	0.0023	316	1	0.08
safetynoCH	408	348	33	0.0156	57	1	0.05
BS151208M	445	242	14	0.052	25	1	0.12
BS201210M	558	307	12	0.0771	25	1	0.17

Continued on next page

Continued from previous page

			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
CLay0305H	275	220	24	0.0169	125	1	0.07
FLay05H	382	342	5499	0.042	5098	1	0.04
FLay06H	566	506	26717	0.0789	30994	1	0.06
RS0805H	720	424	16	0.1488	32	1	0.57
RS0805M02M	360	212	35	0.041	42	1	0.13
RS0805M03M	540	318	12	0.0794	42	1	0.31
RS0805M04M	720	424	16	0.148	32	1	0.58
RS0810H	820	484	28	0.1991	84	1	0.61
RS0810M02M	410	242	61	0.0478	87	1	0.18
RS0810M03M	615	363	34	0.1122	90	1	0.34
RS0810M04M	820	484	28	0.2012	84	1	0.61
RS0815H	940	564	15	0.2328	89	1	0.94
RS0815M02M	470	282	56	0.046	50	1	0.22
RS0815M03M	705	423	83	0.1917	77	1	0.65
RS0815M04M	940	564	15	0.2289	89	1	0.93
RS0820H	1020	604	23	0.3333	98	1	1.47
RS0820M02M	510	302	42	0.0602	103	1	0.25
RS0820M03M	765	453	46	0.1298	119	1	0.55
RS0820M04M	1020	604	30	0.2178	99	1	1.0
RS0830H	1240	744	112	0.3867	733	1	1.1
RS0830M02M	620	372	73	0.0879	290	1	0.29
RS0830M03M	930	558	108	0.2063	535	1	0.61
RS0830M04M	1240	744	111	0.6065	687	1	1.66

Continued on next page

Continued from previous page

			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
RS0830M	250	156	287583	0.0191	53	1	0.03
RS0840H	1440	864	93	0.7896	837	1	2.28
RS0840M02M	720	432	56	0.1168	316	1	0.38
RS0840M03M	1080	648	60	0.4025	275	1	1.32
RS0840M04M	1440	864	93	0.786	837	1	2.25
RS0840M	280	176	143249	0.0328	152	1	0.06
SLay07H	476	392	37	0.039	35	1	0.04
SLay08H	632	520	125	0.0664	121	1	0.08
SLay08M	184	72	36	0.007	35	1	0.01
SLay09H	810	666	236	0.1001	225	1	0.2
SLay10H	1010	830	288	0.1553	280	1	0.27
Syn20M03M	315	195	20	0.0206	108	1	0.07
Syn20M04M	420	260	43	0.0272	215	1	0.12
Syn30M02M	320	200	38	0.0336	114	1	0.14
Syn30M03M	480	300	28	0.0503	163	1	0.18
Syn30M04M	640	400	30	0.1388	221	1	0.48
Syn40M03M	630	390	43	0.0761	185	1	0.34
Syn40M04M	840	520	16	0.1953	291	1	0.79
Safety3	259	161	710	0.004	657	1	0.02
FLay06M	86	26	76932	0.0018	82929	1	0.0
CLay0205H	260	210	11	0.0241	62	1	0.02
CLay0205M	80	30	16	0.0038	84	1	0.0
RS0815M03H	1347	1065	4	0.918	48	1	2.76

Continued on next page

Continued from previous page							
			(NLP(y^k))			NLPR Method	
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time
RS0815M04H	1796	1420	9	0.852	50	1	3.73
RS0820M03H	1467	1155	79	0.1367	57	1	2.4
RS0820M04H	1956	1540	2	0.2967	56	1	4.81
RS0820M	215	131	562489	0.014	57	1	0.02
RS0830M03H	1758	1386	2	0.6667	60	1	5.75
RS0830M04H	2344	1848	2	1.7733	80	1	16.53
RS0840M03H	2040	1608	2	1.1525	84	1	8.04
RS0840M04H	2720	2144	2	1.94	113	1	9.65
SLay07M	140	56	24	0.002	25	1	0.0
SLay09M	234	90	241	0.011	217	1	0.01
SLay10M	290	110	1044	0.015	965	1	0.03
Syn15M04M	340	220	23	0.0382	72	1	0.13
Syn20M02M	210	130	12	0.0096	51	1	0.03
Syn40M	130	90	12	0.0091	59	1	0.03
RS0805M03H	1050	828	6	0.4113	18	1	1.24
RS0810M03H	1185	933	8	0.5244	44	1	1.66
stockcycle	480	48	9785	0.0024	9026	1	0.06

Table A.15: Statistics related to Fixfrac and ECP Methods for Difficult Instances

			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
csched2	400	92	61	0.0	0	4	0.0585	4
fo7	114	72	77	0.0	367	10	0.0094	36
fo72	114	72	93	0.0	469	7	0.0067	28
fo8	146	90	84	0.0	580	8	0.0136	20
fo9	182	110	65	0.0002	290	4	0.012	6
m7	114	72	89	0.0	273	5	0.005	8
o7	114	72	40	0.0	161	8	0.0069	56
o72	114	72	131	0.0001	593	5	0.0054	12
tls12	812	144	151	0.0004	1177	12	0.015	63
tls4	105	16	22	0.0	82	3	0.0	12
tls5	161	25	38	0.0	185	6	0.0005	24
tls6	215	36	73	0.0	423	12	0.0016	71
tls7	345	49	51	0.0	354	12	0.0024	83
trimloss12	812	156	26	0.0008	204	2	0.0284	0
trimloss4	105	20	57	0.0	167	7	0.0004	25
trimloss5	161	30	29	0.0	142	3	0.0016	15
trimloss6	215	42	34	0.0	200	3	0.0017	18
trimloss7	345	56	65	0.0	454	8	0.0023	56
safetynoCH	408	348	37	0.0	70	1	0.0156	0
BS151208M	445	242	6	0.0	6	1	0.052	2
BS201210M	558	307	9	0.0	9	2	0.0771	4
CLay0305H	275	220	105	0.0002	890	11	0.0169	25

Continued on next page

Continued from previous page

			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
FLay05H	382	342	84	0.0	350	5	0.042	6
FLay06H	566	506	37	0.0003	205	3	0.0789	4
RS0805H	720	424	82	0.0	308	9	0.1488	38
RS0805M02M	360	212	90	0.0	149	6	0.041	10
RS0805M03M	540	318	85	0.0	203	6	0.0794	26
RS0805M04M	720	424	82	0.0	308	9	0.148	38
RS0810H	820	484	67	0.0	575	5	0.1991	37
RS0810M02M	410	242	65	0.0002	223	4	0.0478	24
RS0810M03M	615	363	87	0.0001	475	7	0.1122	40
RS0810M04M	820	484	67	0.0001	575	5	0.2012	37
RS0815H	940	564	19	0.0005	277	3	0.2328	30
RS0815M02M	470	282	25	0.0	95	1	0.046	6
RS0815M03M	705	423	28	0.0004	179	3	0.1917	22
RS0815M04M	940	564	19	0.0	277	3	0.2289	30
RS0820H	1020	604	36	0.0	450	7	0.3333	20
RS0820M02M	510	302	28	0.0	137	3	0.0602	23
RS0820M03M	765	453	21	0.0	163	1	0.1298	0
RS0820M04M	1020	604	37	0.0	457	7	0.2178	20
RS0830H	1240	744	21	0.0	166	3	0.3867	30
RS0830M02M	620	372	19	0.0005	147	3	0.0879	22
RS0830M03M	930	558	23	0.0	129	2	0.2063	25
RS0830M04M	1240	744	21	0.0	166	3	0.6065	30
RS0830M	250	156	69	0.0	256	12	0.0191	29

Continued on next page

Continued from previous page

			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
RS0840H	1440	864	37	0.0008	258	2	0.7896	11
RS0840M02M	720	432	29	0.0	168	3	0.1168	15
RS0840M03M	1080	648	8	0.0	95	1	0.4025	3
RS0840M04M	1440	864	37	0.0003	258	2	0.786	11
RS0840M	280	176	44	0.0	112	6	0.0328	12
SLay07H	476	392	15	0.0	0	2	0.039	1
SLay08H	632	520	21	0.0	0	3	0.0664	2
SLay08M	184	72	6	0.0	0	1	0.007	1
SLay09H	810	666	18	0.0	0	2	0.1001	2
SLay10H	1010	830	25	0.0004	0	3	0.1553	3
Syn20M03M	315	195	117	0.0	558	15	0.0206	50
Syn20M04M	420	260	47	0.0	417	4	0.0272	19
Syn30M02M	320	200	103	0.0002	688	9	0.0336	67
Syn30M03M	480	300	19	0.0	281	3	0.0503	63
Syn30M04M	640	400	24	0.0008	489	2	0.1388	36
Syn40M03M	630	390	10	0.0	137	1	0.0761	9
Syn40M04M	840	520	29	0.0007	362	1	0.1953	0
Safety3	259	161	22	0.0	0	1	0.004	1
FLay06M	86	26	28	0.0	161	3	0.0018	10
CLay0205H	260	210	54	0.0002	336	6	0.0241	37
CLay0205M	80	30	72	0.0	537	5	0.0038	23
RS0815M03H	1347	1065	6	0.0017	39	1	0.918	3
RS0815M04H	1796	1420	12	0.0	78	1	0.852	4

Continued on next page

Continued from previous page								
			ECP Method			Fixfrac Method		
Instance	Vars	C.Vars	#NLPs	A.Time	Cuts	#NLPs	A.Time	Cuts
RS0820M03H	1467	1155	8	0.0	65	2	0.1367	4
RS0820M04H	1956	1540	1	0.0	13	1	0.2967	0
RS0820M	215	131	16	0.0	92	2	0.014	7
RS0830M03H	1758	1386	1	0.0	27	1	0.6667	14
RS0830M04H	2344	1848	1	0.0	36	1	1.7733	25
RS0840M03H	2040	1608	2	0.0	44	2	1.1525	21
RS0840M04H	2720	2144	1	0.0	38	1	1.94	25
SLay07M	140	56	7	0.0	0	1	0.002	1
SLay09M	234	90	7	0.0	0	1	0.011	1
SLay10M	290	110	18	0.0	0	1	0.015	1
Syn15M04M	340	220	59	0.0	370	5	0.0382	21
Syn20M02M	210	130	79	0.0001	305	11	0.0096	22
Syn40M	130	90	114	0.0	349	10	0.0091	25
RS0805M03H	1050	828	7	0.0	33	2	0.4113	6
RS0810M03H	1185	933	6	0.0017	57	1	0.5244	2
stockcycle	480	48	5	0.0	0	1	0.0024	1

For Tables A.16–A.18, the row-headings are described as follows:

- Total: Total time spent for the instance.
- LP: Total time spent in solving LPs.
- NLP: Total time spent in solving NLPs.

- Heuristic: Total time spent in primal heuristics.
- Cut Gen: Total Time spent in generating linearizations.
- Cut Pool: Total time spent in searching violated cuts from MINTO's cut pool.

Table A.16: Statistics of Time Spent for Easy Instances

Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
alan	0.0	0.0	0.0	0.0	0.0	0.0
batch	0.06	0.04	0.0	0.0	0.0	0.0
batchdes	0.0	0.0	0.0	0.0	0.0	0.0
csched1	0.11	0.02	0.02	0.0	0.0	0.0
enpro48	0.76	0.42	0.11	0.02	0.07	0.0
enpro56	0.91	0.55	0.08	0.01	0.09	0.0
ex1223	0.0	0.0	0.0	0.0	0.0	0.0
ex1223a	0.0	0.0	0.0	0.0	0.0	0.0
ex1223b	0.0	0.0	0.0	0.0	0.0	0.0
ex3	0.0	0.0	0.0	0.0	0.0	0.0
ex4	0.32	0.18	0.02	0.03	0.02	0.0
fac1	0.02	0.0	0.0	0.0	0.0	0.0
fac2	0.07	0.01	0.02	0.0	0.01	0.0
fac3	0.06	0.02	0.01	0.0	0.03	0.0
gbd	0.0	0.0	0.0	0.0	0.0	0.0
m3	0.1	0.06	0.0	0.0	0.01	0.0
meanvarx	0.01	0.0	0.0	0.0	0.0	0.0
optprloc	0.22	0.12	0.01	0.01	0.02	0.0
prob02	0.0	0.0	0.0	0.0	0.0	0.0

Continued on next page

Continued from previous page

Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
prob03	0.0	0.0	0.0	0.0	0.0	0.0
ravem	0.23	0.1	0.02	0.01	0.01	0.0
ste14	0.01	0.01	0.0	0.0	0.0	0.0
ste38	0.0	0.0	0.0	0.0	0.0	0.0
stmiqp1	0.0	0.0	0.0	0.0	0.0	0.0
stmiqp2	0.0	0.0	0.0	0.0	0.0	0.0
stmiqp3	0.01	0.0	0.0	0.0	0.0	0.0
stmiqp4	0.0	0.0	0.0	0.0	0.0	0.0
stmiqp5	0.0	0.0	0.0	0.0	0.0	0.0
sttest1	0.0	0.0	0.0	0.0	0.0	0.0
sttest2	0.0	0.0	0.0	0.0	0.0	0.0
sttest3	0.0	0.0	0.0	0.0	0.0	0.0
sttest4	0.0	0.0	0.0	0.0	0.0	0.0
sttest5	0.01	0.01	0.0	0.0	0.0	0.0
sttest6	0.02	0.0	0.0	0.0	0.02	0.0
sttest8	0.0	0.0	0.0	0.0	0.0	0.0
sttestgr1	0.62	0.38	0.01	0.0	0.06	0.0
sttestgr3	0.15	0.08	0.0	0.01	0.03	0.0
sttestph4	0.0	0.0	0.0	0.0	0.0	0.0
synthes1-10	0.01	0.01	0.0	0.0	0.0	0.0
synthes1	0.01	0.0	0.0	0.0	0.0	0.0
synthes2	0.02	0.01	0.0	0.0	0.0	0.0
synthes3	0.03	0.01	0.0	0.0	0.0	0.0
tls2	0.19	0.12	0.0	0.01	0.01	0.0

Continued on next page

Continued from previous page

Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
trimloss2	0.18	0.11	0.0	0.01	0.01	0.0
BNS121612	0.81	0.38	0.04	0.02	0.03	0.0
BNS162010	3.17	2.26	0.1	0.03	0.09	0.0
BNS8125	0.21	0.08	0.05	0.01	0.01	0.0
BSBM8664	2.32	1.95	0.13	0.02	0.04	0.0
BSCH8664	5.33	4.03	0.89	0.05	0.2	0.0
Process20M	0.46	0.29	0.01	0.01	0.05	0.0
Process8CH	0.02	0.0	0.01	0.0	0.01	0.0
Process8C	0.01	0.0	0.01	0.0	0.01	0.0
SLBM42	0.78	0.6	0.03	0.01	0.02	0.0
CLay0203H	1.17	0.96	0.07	0.02	0.01	0.0
CLay0203M	0.36	0.27	0.02	0.0	0.01	0.0
CLay0204H	5.11	4.28	0.15	0.05	0.17	0.0
CLay0204M	1.12	0.83	0.02	0.0	0.11	0.0
CLay0303H	4.41	3.63	0.14	0.06	0.07	0.02
CLay0303M	0.72	0.51	0.01	0.01	0.02	0.0
FLay02H	0.01	0.0	0.0	0.0	0.0	0.0
FLay02M	0.0	0.0	0.0	0.0	0.0	0.0
FLay03H	0.43	0.29	0.09	0.0	0.01	0.0
FLay03M	0.15	0.1	0.01	0.0	0.02	0.0
FLay04M	4.86	4.04	0.09	0.04	0.13	0.02
SLay04H	0.32	0.19	0.04	0.01	0.01	0.0
SLay04M	0.12	0.07	0.0	0.01	0.0	0.0
SLay05H	1.1	0.77	0.11	0.03	0.03	0.0

Continued on next page

Continued from previous page

Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
SLay05M	0.19	0.1	0.02	0.0	0.01	0.0
Syn05H	0.01	0.0	0.0	0.0	0.0	0.0
Syn05M02H	0.1	0.01	0.07	0.01	0.07	0.0
Syn05M02M	0.07	0.02	0.02	0.01	0.01	0.0
Syn05M03H	0.16	0.01	0.14	0.0	0.12	0.0
Syn05M03M	0.19	0.08	0.03	0.01	0.01	0.0
Syn05M04H	0.36	0.0	0.33	0.01	0.28	0.0
Syn05M04M	0.33	0.13	0.07	0.01	0.05	0.0
Syn05M	0.03	0.03	0.0	0.0	0.0	0.0
Syn10H	0.01	0.01	0.0	0.0	0.0	0.0
Syn10M02H	0.25	0.01	0.21	0.0	0.16	0.0
Syn10M02M	1.01	0.72	0.03	0.01	0.04	0.01
Syn10M03H	0.51	0.02	0.46	0.01	0.35	0.0
Syn10M04H	1.19	0.05	1.08	0.01	0.83	0.0
Syn10M	0.05	0.02	0.0	0.0	0.0	0.0
Syn15H	0.05	0.0	0.04	0.0	0.01	0.0
Syn15M02H	0.75	0.01	0.72	0.0	0.58	0.0
Syn15M03H	1.48	0.02	1.4	0.01	1.19	0.0
Syn15M04H	2.79	0.05	2.68	0.0	2.1	0.0
Syn15M	0.16	0.06	0.02	0.0	0.03	0.0
Syn20H	0.1	0.01	0.07	0.0	0.04	0.0
Syn20M02H	1.16	0.02	1.06	0.01	0.79	0.0
Syn20M03H	2.13	0.06	1.94	0.01	1.6	0.0
Syn20M	0.95	0.67	0.05	0.01	0.05	0.0

Continued on next page

Continued from previous page

Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
Syn30H	0.5	0.03	0.42	0.0	0.33	0.0
Syn30M02H	2.22	0.06	2.06	0.01	1.34	0.0
Syn40H	0.76	0.04	0.62	0.01	0.44	0.0
Water0202R	0.85	0.11	0.65	0.02	0.23	0.0
Water0303R	43.94	21.1	20.83	0.82	2.02	0.01
Syn40M04H	8.03	1.95	4.34	0.13	2.49	0.0

Table A.17: Statistics of Time Spent for Moderate Instances

Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
m6	36.59	31.47	0.04	0.35	0.74	0.08
BSBM9964	46.01	42.44	0.35	1.15	0.31	0.01
BSCH9964	78.67	70.22	3.37	0.82	1.25	0.01
BS10BM1064	540.02	504.62	1.8	5.95	1.7	0.44
BSBM1064	887.15	833.07	2.05	9.73	2.09	0.8
SafetynoBM	36.41	29.05	0.11	0.65	0.83	0.04
BS101006M	15.24	13.38	0.31	0.2	0.12	0.0
BS121208M	36.69	32.43	0.55	0.94	0.22	0.02
FLay05M	823.72	735.62	8.58	17.23	3.61	7.94
CLay0304H	30.19	25.5	0.65	0.45	0.38	0.17
CLay0304M	20.6	17.14	0.09	0.2	0.38	0.19
CLay0305M	43.75	35.38	0.16	0.92	0.71	0.48
RSyn0810M	13.9	10.42	0.1	0.24	0.5	0.01

Continued on next page

Continued from previous page						
Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
RSyn0815M	15.99	12.11	0.14	0.31	0.42	0.0
SLay06H	10.22	8.23	0.3	0.35	0.17	0.0
Syn15M03M	4.16	2.8	0.3	0.02	0.23	0.0
Syn30M	6.3	5.02	0.06	0.09	0.12	0.05
SL4Convex	5.75	4.72	0.15	0.11	0.18	0.0
SLBM52	1.71	1.18	0.06	0.0	0.11	0.01
FLay04H	13.66	11.21	1.01	0.2	0.29	0.06
RS0805M02H	3.11	0.44	1.99	0.06	1.13	0.0
RS0805M04H	7.67	0.41	6.19	0.19	3.32	0.0
RS0810M02H	3.86	0.54	2.54	0.06	1.51	0.0
RS0815M02H	3.14	0.4	2.05	0.09	1.07	0.0
RS0820M02H	4.55	0.48	3.17	0.09	1.31	0.0
RS0830M02H	5.61	0.11	5.32	0.12	4.93	0.0
RS0840M02H	10.74	0.57	9.42	0.09	3.78	0.0
SLay06M	0.99	0.66	0.04	0.01	0.06	0.0
Syn10M03M	3.15	2.31	0.12	0.06	0.09	0.01
Syn10M04M	7.12	5.42	0.19	0.1	0.19	0.02
Syn15M02M	1.21	0.78	0.08	0.02	0.07	0.0
Syn20M04H	3.29	0.04	3.21	0.01	2.85	0.0
Syn30M03H	3.65	0.05	3.54	0.03	2.91	0.0
Syn30M04H	8.97	0.28	8.24	0.09	6.28	0.0
Syn40M02H	2.12	0.27	1.49	0.02	1.0	0.0
Syn40M03H	9.53	0.69	8.09	0.06	4.85	0.0
RS0810M04H	15.09	0.96	12.41	0.25	4.25	0.0

Table A.18: Statistics of Time Spent for Difficult Instances

Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
csched2	960.85	794.85	46.73	7.4	7.94	8.18
fo7	2237.83	1937.71	0.18	34.51	36.44	16.08
fo72	813.49	708.87	0.08	13.56	11.08	4.59
fo8	2866.85	2490.11	0.17	57.48	31.23	20.3
fo9	8175.64	6976.94	0.22	43.9	153.81	66.28
m7	173.46	149.55	0.05	1.56	3.49	0.82
o7	4145.56	3553.14	0.12	41.08	128.35	13.24
o72	5542.94	4703.61	0.16	81.95	101.18	63.15
tls12	14405.42	13298.26	0.75	353.65	2.22	109.98
tls4	6.77	5.36	0.01	0.11	0.05	0.04
tls5	8695.77	7720.15	0.11	84.28	18.26	99.49
tls6	14400.17	12629.05	0.33	237.62	31.73	126.23
tls7	14400.13	11661.22	0.33	522.31	61.4	205.24
trimloss12	14400.14	13107.57	1.91	523.63	6.19	83.16
trimloss4	903.38	806.2	0.05	6.21	4.74	6.06
trimloss5	14400.07	12490.23	0.45	190.47	46.89	158.02
trimloss6	14400.1	12575.94	0.62	280.93	39.39	164.55
trimloss7	14400.11	12004.52	0.26	516.73	44.04	174.57
safetynoCH	736.62	641.63	0.58	9.61	6.53	0.78
BS151208M	65.45	60.01	0.9	0.6	0.36	0.01
BS201210M	222.86	207.7	1.25	1.21	0.72	0.03
CLay0305H	129.42	112.17	0.68	0.23	1.33	0.94

Continued on next page

Continued from previous page

Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
FLay05H	2031.85	1649.21	231.01	31.48	4.97	17.58
FLay06H	14400.01	11071.89	2108.94	230.51	52.31	195.1
RS0805H	706.23	593.64	4.29	16.86	4.81	2.25
RS0805M02M	127.12	101.74	1.81	2.77	1.64	0.25
RS0805M03M	486.37	419.93	1.74	10.1	2.8	0.96
RS0805M04M	707.34	594.52	4.28	16.84	4.89	2.2
RS0810H	8295.79	6701.99	7.18	268.12	54.41	61.61
RS0810M02M	2094.2	1643.26	3.3	53.34	21.31	8.95
RS0810M03M	7142.1	5955.56	4.95	156.04	77.14	56.29
RS0810M04M	8201.34	6628.51	7.26	264.7	52.2	59.85
RS0815H	10447.67	8031.27	5.14	314.27	116.4	33.1
RS0815M02M	4631.8	3463.41	2.84	124.42	110.52	17.78
RS0815M03M	9267.57	6786.67	17.15	259.93	150.21	120.88
RS0815M04M	10735.02	8229.69	5.05	323.26	134.13	45.56
RS0820H	14400.11	10480.83	11.47	496.11	183.19	142.6
RS0820M02M	4540.48	3490.31	2.96	114.44	46.82	16.28
RS0820M03M	8034.41	6169.62	6.65	144.14	117.12	41.03
RS0820M04M	13776.2	10417.69	9.06	424.68	194.36	93.74
RS0830H	14400.13	10050.34	45.57	649.93	173.51	241.79
RS0830M02M	6860.15	4728.82	6.98	241.15	194.99	83.33
RS0830M03M	10212.26	7729.86	23.3	221.28	132.85	131.23
RS0830M04M	14400.21	9659.51	70.8	762.18	141.86	308.44
RS0830M	14400.02	5600.12	5483.36	38.21	446.88	503.66
RS0840H	14400.03	10573.58	77.32	289.94	107.68	235.05

Continued on next page

Continued from previous page

Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
RS0840M02M	8229.78	5864.45	7.27	262.5	198.46	85.61
RS0840M03M	14066.99	10415.35	25.87	333.11	155.36	129.59
RS0840M04M	14400.03	10582.77	76.93	288.71	108.14	237.13
RS0840M	14400.02	6338.47	4701.08	55.43	377.95	427.01
SLay07H	60.05	50.75	1.56	0.71	0.6	0.04
SLay08H	553.5	480.13	8.58	10.55	2.97	1.29
SLay08M	8.72	6.45	0.27	0.2	0.17	0.01
SLay09H	2993.57	2558.62	24.03	69.77	11.22	16.29
SLay10H	14400.0	12417.42	45.48	155.79	69.25	78.06
Syn20M03M	334.68	252.79	0.79	5.47	4.38	1.55
Syn20M04M	2326.24	1754.38	1.4	36.01	22.45	13.46
Syn30M02M	1486.62	1135.46	1.74	24.67	17.85	23.3
Syn30M03M	4474.48	3477.37	1.74	72.98	39.25	30.01
Syn30M04M	9217.77	6774.98	4.94	154.92	108.3	170.58
Syn40M03M	4974.87	3758.72	3.69	72.66	37.97	17.62
Syn40M04M	10862.58	8104.3	4.13	163.7	92.89	125.11
Safety3	5515.32	4631.91	2.84	128.05	59.42	62.34
FLay06M	11703.2	10320.15	141.63	252.04	65.84	146.41
CLay0205H	70.02	58.65	0.44	1.69	0.94	0.2
CLay0205M	18.81	15.13	0.08	0.06	0.42	0.22
RS0815M03H	11.34	1.83	7.36	0.3	3.89	0.0
RS0815M04H	16.26	1.87	12.25	0.27	5.05	0.0
RS0820M03H	20.08	3.82	13.47	0.32	3.35	0.01
RS0820M04H	6.38	0.18	5.7	0.35	4.95	0.0

Continued on next page

Continued from previous page						
Instance	Total	LP	NLP	Heuristic	Cut Gen	Cut Pool
RS0820M	14400.02	3638.08	7889.86	18.02	469.8	734.09
RS0830M03H	8.2	0.16	7.75	0.2	7.14	0.0
RS0830M04H	22.93	0.36	21.85	0.52	20.13	0.0
RS0840M03H	13.21	0.16	12.65	0.25	11.11	0.0
RS0840M04H	16.43	0.28	15.47	0.46	12.64	0.0
SLay07M	2.04	1.44	0.05	0.03	0.09	0.0
SLay09M	481.76	399.47	2.67	9.74	3.65	4.27
SLay10M	14400.01	11514.09	15.71	429.23	103.78	490.7
Syn15M04M	24.3	17.4	1.2	0.35	0.64	0.18
Syn20M02M	26.78	19.99	0.26	0.31	0.65	0.08
Syn40M	84.99	64.36	0.23	1.13	2.44	0.29
RS0805M03H	6.15	0.53	4.53	0.13	2.3	0.0
RS0810M03H	9.38	1.4	6.39	0.14	2.37	0.0
stockcycle	4434.7	3097.8	23.34	112.6	104.86	195.85

Bibliography

K. Aardal, G. L. Nemhauser, and R. Weismantel, editors. *Discrete Optimization, Handbook in Operations Research and Management Science*, volume 12. Elsevier, 2005.

M. A. Abramson. Mixed variable optimization of a load-bearing thermal insulation system using a filter pattern search algorithm. *Optimization and Engineering*, 5: 157–177, 2004.

M. A. Abramson, C. Audet, and J. J. E. Dennis. Filter pattern search algorithms for mixed variable constrained optimization problems. Technical Report TR04-09, CAAM, Rice University, 2004.

T. Achterberg and T. Berthold. Improving the feasibility pump. ZIB-Report 05-42, Konrad-Zuse Zentrum für Informationstechnik Berlin, Berlin, Germany, September 2005.

T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2004.

I. Akrotirianakis, I. Maros, and B. Rustem. An outer approximation based branch-and-cut algorithm for convex 0-1 MINLP problems. *Optimization Methods and Software*, 16:21–47, 2001.

G. Allaire and C. Castro. A new approach for the optimal distribution of assemblies in a nuclear reactor. *Numerische Mathematik*, 89(1):1–29, 2001.

A. Atamtürk. Cover and pack inequalities for (mixed) integer programming. *Annals of Operations Research*, 2004. forthcoming.

A. Atamtürk and V. Narayanan. Conic mixed integer rounding cuts. In *IPCO 2007: The Twelfth Conference on Integer Programming and Combinatorial Optimization*, pages 16–29. Springer, 2007.

A. Atamtürk and M. Savelsbergh. Integer-programming software systems. *Annals of Operations Research*, 140:67–124, 2005.

C. Audet and J. Dennis. A pattern search filter method for nonlinear programming without derivatives. *SIAM J. Optimization*, 14(4):980–1010, 2004.

C. Audet and J. E. Dennis. Pattern search algorithms for mixed variable programming. *SIAM J. on Optimization*, 11(3):573–594, 2000. ISSN 1052-6234.

E. Balas. Duality in discrete programming: Ii. the quadratic case. *Management Science*, 16(1):14–32, 1969.

E. Balas. Disjunctive programming. In *Annals of Discrete Mathematics 5: Discrete Optimization*, pages 3–51. North Holland, 1979.

E. Balas and J. Mazzola. Nonlinear 0-1 programming: I. Linearization techniques. *Mathematical Programming*, 30:1–21, 1984a.

E. Balas and J. Mazzola. Nonlinear 0-1 programming: II. Dominance relations and algorithms. *Mathematical Programming*, 30:22–45, 1984b.

E. Balas, S. Ceria, and G. Corneujols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324, 1993.

E. Balas, S. Ceria, and G. Cornuejols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42:1229–1246, 1996.

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch and price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley and Sons, New York, second edition, 1993.

J. M. Beal, A. Shukla, O. A. Brezhneva, and M. A. Abramson. Optimal sensor placement for enhancing sensitivity to change in stiffness for structural health monitoring. Technical report, Department of Mathematics and Statistics, Air Force Institute of Technology, 2006. URL `http://www.afit.edu/en/ENC/Faculty/MAbramson/NOMADm.html`.

J. F. Benders. Partitioning procedures for solving mixed variable programming problems. *Numerische Mathematik*, 4:238–252, 1962.

M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed–integer linear programming. *Mathematical Programming*, 1: 76–94, 1971.

L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 2006. to appear.

D. Bertsimas and R. Weismantel, editors. *Optimization over Integers*. Dynamic Ideas, 2005.

D. Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74:121–140, 1996.

P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. Research Report RC23862 (W0602-029), IBM, 2006.

P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 2008. to appear.

B. Borchers and J. E. Mitchell. An improved branch and bound algorithm for mixed integer nonlinear programs. *Computers & Operations Research*, 21:359–368, 1994.

B. Borchers and J. E. Mitchell. A computational comparison of branch and bound and outer approximation algorithms for 0-1 mixed integer nonlinear programs. *Computers & Operations Research*, 24:699–701, 1997.

M. R. Bussieck, A. S. Drud, and A.Meeraus. MINLPLib - a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15 (1), 2003.

I. Castillo, J. Westerlund, S. Emet, and T. Westerlund. Optimization of block layout deisgn problems with unequal areas: A comparison of milp and minlp optimization methods. *Computers and Chemical Engineering*, 30:54–69, 2005.

M. T. Cezik and G. Iyengar. Cuts for mixed 0-1 conic programming. *Mathematical Programming*, 104:179–202, 2005.

COIN-OR. Computational Infrastructure for Operations Research, 2004. `http://www.coin-or.org`.

G. Cornuejols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112(1):3–44, 2008.

CPLEX Optimization. *Using the CPLEX Callable Library, Version 9*. CPLEX Optimization, Inc., Incline Village, NV, 2005.

H. Crowder, E. L. Johnson, and M. W. Padberg. Solving large scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

R. J. Dakin. A tree search algorithm for mixed programming problems. *Computer Journal*, 8:250–255, 1965.

E. Danna, E. Rothberg, and C. LePape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102:71–90, 2005.

D. De Wolf and Y. Smeers. The gas transmission problem solved by an extension of the simplex algorithm. *Management Science*, 46:1454–1465, 2000.

R. S. Dembo, J. M. Mulvey, and S. A. Zenios. Large-scale nonlinear network models and their applications. *Operations Research*, 37:353–372, 1989.

E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

M. A. Duran and I. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2002.

M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2005.

R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

R. Fletcher and S. Leyffer. Numerical experience with lower bounds for MIQP branch-and-bound. *SIAM Journal on Optimization*, 8:604–616, 1998a.

R. Fletcher and S. Leyffer. User manual for filterSQP, 1998b. University of Dundee Numerical Analysis Report NA-181.

R. Fletcher, S. Leyffer, and P. Toint. On the global convergence of a Filter-SQP algorithm. *SIAM Journal on Optimization*, 13:44–59, 2002.

O. E. Flippo and A. H. G. R. Kan. A note on benders decomposition in mixed-integer quadratic programming. *Operations Research Letters*, 9:81–83, 1990.

O. E. Flippo and A. H. G. R. Kan. Decomposition in general mathematical programming. *Mathematical Programming*, 60:361–382, 1993.

C. Floudas. *Nonlinear and Mixed–Integer Optimization*. Topics in Chemical Engineering. Oxford University Press, New York, 1995.

J. Forrest. CBC, 2004. Available from `http://www.coin-or.org/`.

J. J. H. Forrest, J. P. H. Hirst, and J. A. Tomlin. Practical solution of large scale mixed integer programming problems with UMPIRE. *Management Science*, 20: 736–773, 1974.

R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press, 1993.

A. Frangioni and C. Gentile. Perspective cuts for a class of convex 0-1 mixed integer programs. *Mathematical Programming*, 106:225–236, 2006.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

R. S. Garfinkel and G. L. Nemhauser. *Integer Programming*. John Wiley, New York, 1972.

D. M. Gay. Hooking your solver to AMPL. Technical Report 97-4-06, Computing Sciences Research Center, Bell Laboratories, 1997.

A. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, 1972.

R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Monthly*, 64:275–278, 1958.

J.-P. Goux and S. Leyffer. Solving large MINLPs on computational grids. *Optimization and Engineering*, 3:327–354, 2003.

J.-P. Goux, S. Kulkarni, J. Linderoth, and M. Yoder. An enabling framework for master-worker applications on the computational grid. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, pages 43–50, Los Alamitos, CA, 2000a. IEEE Computer Society.

J.-P. Goux, J. T. Linderoth, and M. E. Yoder. Metacomputing and the master-worker

paradigm. Preprint ANL/MCS-P792-0200, Mathematics and Computer Science Division, Argonne National Laboratory, 2000b.

D. Granot, F. Granot, and W. Vaessen. An accelerated covering relaxation algorithm for solving 0-1 positive polynomial programs. *Mathematical Programming*, 22:350–357, 1982.

I. E. Grossmann. Review of nonlinear mixed–integer and disjunctive programming techniques. *Optimization and Engineering*, 3:227–252, 2002.

I. E. Grossmann and R. W. H. Sargent. Optimal design of multipurpose batch plants. *Industrial & Engineering Chemistry Process Design and Development*, 18:343–348, 1979.

O. K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

I. Harjunkoski, T. Westerlund, R. Pörn, and H. Skrifvars. Different transformations for solving non–convex trim loss problems by MINLP. *European J. Operational Research*, 105:594–603, 1988.

V. Jain and I. E. Grossmann. Cyclic scheduling of continuous parallel-process units with decaying performance. *AIChE Journal*, 44(7):1623–1636, 1998.

E. Johnson, G. Nemhauser, and M. Savelsbergh. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing*, 12:2–23, 2000.

J. E. Kelley. The cutting plane method for solving convex programs. *Journal of SIAM*, 8(4):703–712, 1960.

G. R. Kocis and I. E. Grossmann. Relaxation strategy for the structural optimization of process flowheets. *Industrial Engineering Chemical Research*, 26:1869–1880, 1987.

G. R. Kocis and I. E. Grossmann. Global optimization of nonconvex mixed–integer nonlinear programming (MINLP) problems in process synthesis. *Industrial Engineering Chemistry Research*, 27:1407–1421, 1988.

G. R. Kocis and I. E. Grossmann. Computational experience with DICOPT solving MINLP problems in process systems engineering. *Computers and Chemical Engineering*, 13:307–315, 1989.

M. Kokkolaras, C. Audet, and J. E. Dennis. Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system. *Optimization and Engineering*, 2:5–29, 2001.

A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

R. Laundy. Implementation of parallel branch and bound algorithms in XPRESS-MP. Dash Internal Report no. DA/TR-101, January 1998.

S. Leyffer. Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Comput. Optim. Appl.*, 18(3):295–309, 2001.

S. Leyffer. MacMINLP: Test problems for mixed integer nonlinear programming, 2003. `http://www-unix.mcs.anl.gov/~leyffer/macminlp`.

S. Leyffer. *Deterministic Methods for Mixed Integer Nonlinear Programming*. PhD thesis, University of Dundee, Dundee, Scotland, UK, 1993.

S. Leyffer. User manual for MINLP-BB, 1998. University of Dundee.

J. Linderoth and T. Ralphs. Noncommercial software for mixed-integer linear programming. In *Integer Programming: Theory and Practice*, pages 253–303. CRC Press Operations Research Series, 2005.

J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies in mixed integer programming. *INFORMS Journal on Computing*, 11:173–187, 1999.

M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP*, 11, 1997.

L. Lovász and A. Schrijver. Cones of matrices and setfunctions, and 0-1 optimization. *SIAM Journal on Optimization*, 1, 1991.

H. Marchand and L. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49:363–371, 2001.

H. Marchand and L. Wolsey. The 0-1 knapsack problem with a single continuous variable. *Mathematical Programming*, 85:15–33, 1999.

H. Marchand, A. Martin, R. Weismantel, and L. Wolsey. Cutting planes in integer and mixed integer programming, 1999. DP 9953, CORE, Universite catholique de Louvainla -Neuve.

R. D. McBride and J. S. Yormark. An implicit enumeration algorithm for quadratic integer programming. *Management Science*, 26(3):282–296, March 1980.

P. Michelon and N. Maculan. Lagrangean decomposition for integer nonlinear programming with linear constraints. *Mathematical Programming*, 52:303–313, 1991.

S. V. Nabar and L. Schrage. Modeling and solving nonlinear integer programming problems. Technical report, Annual AIChE Meeting, Chicago (IL), 1990. paper No. 22a.

G. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York, 1988.

G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47–58, 1994.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

M. Padberg, T. J. Van Roy, and L. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33:842–861, 1985.

M. W. Padberg. A note on 0-1 programming. *Operations Research*, 23:833–837, 1979.

P. M. Pardalos and S. A. Vavasis. Quadratic programming with one negative eigenvalue is np-hard. *Journal of Global Optimization*, 1:15–22, 1992.

I. Quesada and I. E. Grossmann. An LP/NLP based branch–and–bound algorithm for convex MINLP optimization problems. *Computers and Chemical Engineering*, 16:937–947, 1992.

A. J. Quist, R. van Gemeert, J. E. Hoogenboom, T. Ílles, C. Roos, and T. Terlaky. Application of nonlinear optimization to reactor core fuel reloading. *Annals of Nuclear Energy*, 26:423–448, 1998.

T. Ralphs and L. Ladányi. SYMPHONY : A parallel framework for branch, cut,

and price. Available from ftp://ftp.branchandcut.org/pub/reference/symphony.ps, 2000.

N. V. Sahinidis and I. E. Grossmann. Convergence properties of generalized benders decomposition. *Computers and Chemical Engineering*, 15:481–491, 1991.

M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

N. W. Sawaya, C. D. Laird, and P. Bonami. A novel library of non-linear mixed-integer and generalized disjunctive programming problems. In preparation, 2006.

A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, 1986.

G. B. Sheble and G. N. Fahd. Unit commitment literature synopsis. *IEEE Transactions on Power Systems*, 9:128–135, 1994.

H. D. Sherali and W. P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM J. Discrete Math.*, 3:411–430, 1990.

H. D. Sherali and W. P. Adams. A hierarchy of relaxations and convex hull characterizations for mixed-integer zero-one programming problems. *Discrete Applied Math.*, 52:83–106, 1994.

R. Stubbs and S. Mehrohtra. Generating convex polynomial inequalities for mixed 0-1 programs. *Journal of Global Optimization*, 24:311–332, 2002.

R. Stubbs and S. Mehrotra. A branch-and-cut method for 0-1 mixed convex programming. *Mathematical Programming*, 86:515–532, 1999.

S. A. Vavasis. *Nonlinear optimization: complexity issues*. Oxford University Press, Inc., New York, NY, USA, 1991. ISBN 0-19-507208-1.

J. Viswanathan and I. E. Grossmann. A combined penalty function and outer–approximation method for MINLP optimization. *Computers and Chemical Engineering*, 14:769–782, 1990.

J. Viswanathan and I. E. Grossmann. Optimal feed location and number of trays for distillation columns with multiple feeds. *I&EC Research*, 32:2942–2949, 1993.

A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

T. Westerlund and F. Pettersson. An extended cutting plane method for solving convex MINLP problems. *Computers and Chemical Engineering*, 19(Supplement 1):131–136, 1995.

T. Westerlund and R. Pörn. Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. *Optimization and Engineering*, 3(3):253–280, 2002.

T. Westerlund, F. Pettersson, and I. E. Grossmann. Optimization of pump configurations as MINLP problem. *Computers and Chemical Engineering*, 18(9):845–858, 1994.

T. Westerlund, H. Skrifvars, I. Harjunkoski, and R. Pörn. An extended cutting plane method for a class of non-convex minlp problems. *Computers and Chemical Engineering*, 22(3):357–365, 1998.

L. A. Wolsey. *Integer Programming*. John Wiley and Sons, New York, 1998.

Z. Zhao, J. Meza, and M. van Hove. Using pattern search methods for surface structure determination of nanostructures. Technical Report LBNL-57541, Lawrence Berkeley National Laboratory, 2005.

Curriculum Vita

Name: Kumar Abhishek

Date of Birth: March 28th, 1979.

Place of Birth: Rourkela, India.

Parents: Mrs. Shakuntala Singh and Mr. Bhairava Nath Singh.

Education

- Doctoral Candidate in Industrial Engineering August, 2003 – present.
  Lehigh University, Bethlehem, PA.
- M.S.in Management Science August, 2003 – Jan, 2008.
  Lehigh University, Bethlehem, PA.
- B.Tech. in Industrial Engineering July, 1997 – June, 2001.
  Indian Institute of Technology (IIT), Kharagpur, India.

Publications

- K. Abhishek and S. Leyffer and J. T. Linderoth. Modeling without Categorical Variables: A Mixed Integer Nonlinear Program for the Optimization of Thermal

Insulation Systems, Submitted for publication in Optimization and Engineering.

- K. Abhishek and S. Leyffer and J. T. Linderoth. FiLMINT: An Outer Approximation based Solver for Mixed Integer Nonlinear Programs, Submitted for publication in INFORMS Journal of Computing.

- K. Abhishek and S. Leyffer and J. T. Linderoth. Branch-and-Pump Heuristics for Mixed Integer Nonlinear Programming, Draft in preparation.

Conference Presentations

- K. Abhishek and S. Leyffer and J. T. Linderoth. FiLMINT: An Outer Approximation based Solver for Mixed Integer Nonlinear Programs, INFORMS Annual Meeting, Seattle, WA., November 2007.

- K. Abhishek and S. Leyffer and J. T. Linderoth. Heuristics for Mixed Integer Nonlinear Programs, INFORMS Annual Meeting, Seattle, WA., November 2007.

- K. Abhishek and S. Leyffer and J. T. Linderoth. FiLMINT: A Linearizations based MINLP solver and the Feasibility Pump, ICCOPT II, McMaster University, Hamilton, ON, Canada, August, 2007.

- K. Abhishek and S. Leyffer and J. T. Linderoth. FiLMINT: A Linearizations based MINLP solver, ACNW Optimization Tutorials, Gleacher Center, Chicago, IL, June 2007.

- K. Abhishek and S. Leyffer and J. T. Linderoth. FiLMINT: An Outer Approximation based Solver for Mixed Integer Nonlinear Programs, INFORMS Annual Meeting, Pittsburgh, PA., November 2006.

- K. Abhishek and S. Leyffer and J. T. Linderoth. FiLMINT: An Outer Approximation based Solver for Mixed Integer Nonlinear Programs, MIP 2006: Workshop on Mixed Integer Programming, Miami, FL., June 2006.

Experience

- RCEAS Dean Doctoral Scholar, August, 2007 – April, 2008.
  Lehigh University.

- Givens Associate, June, 2007 – August, 2007
  MCS Division, Argonne National Laboratory, IL.

- Research Assistant, August, 2006 – May, 2007
  Department of Industrial Engineering, Lehigh University.

- Givens Associate, June, 2006 – August, 2006.
  MCS Division, Argonne National Laboratory, IL.

- Research Assistant, August, 2004 – May, 2006.
  Department of Industrial Engineering, Lehigh University.

- Value Chain Research Team Member, Sept., 2003 – July, 2004.
  Department of Industrial Engineering, Lehigh University.

- Research Consultant, July. 2002 – July. 2003.
  SRIC, Indian Institute of Technology, Kharagpur, India.

- System Analyst, March 2002 – June. 2002.
  P & O Nedlloyd, Pune, India.

- Software Engineer, July 2001 – Feb. 2002.

  IBM Global Services, Pune, India.

Honors and Activities

- Awarded the RCEAS Dean's Doctoral Assistantship for the year 2007-2008 by Lehigh University.
- Awarded the Gotshall Fellowship for the year 2003-2004 by the Dept. of ISE, Lehigh University.
- Member of Phi Beta Delta International Honorary Society.
- Member of INFORMS Student Chapter, Lehigh University.
- Member of Mathematical Programming Society.
- President of Lehigh University Cricket Club for 2007-2008.
- Vice Captain of Lehigh University Cricket Club for 2006-2008.