GLOBAL OPTIMIZATION OF NONCONVEX NONLINEAR PROGRAMS USING PARALLEL BRANCH AND BOUND

By

Thomas Guthrie Weidner Epperly

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

(Chemical Engineering)

at the

UNIVERSITY OF WISCONSIN – MADISON

1995

Abstract

A branch and bound algorithm for computing globally optimal solutions to non-convex nonlinear programs in continuous variables is presented. The algorithm is directly suitable for a wide class of problems arising in chemical engineering design. It can solve problems defined using algebraic functions and twice differentiable transcendental functions, in which finite upper and lower bounds can be placed on each variable. The algorithm uses rectangular partitions of the variable domain and a new bounding program based on convex/concave envelopes and positive definite combinations of quadratic terms. The algorithm is deterministic and obtains convergence with final regions of finite size. The partitioning strategy uses a sensitivity analysis of the bounding program to predict the best variable to split and the split location. Two versions of the algorithm are considered, the first using a local NLP algorithm (MINOS) and the second using a sequence of lower bounding programs in the search for local minima. Runtime results for both versions are presented for a set of 50 test problems. In addition, a parallel version of the branch and bound algorithm is presented and tested. Parallelism is achieved by applying a separate processor to the bounding of each region. Each processor maintains a list of regions to be analyzed, and a load balancing scheme is used to balance the quality and quantity of regions available in each processor's list.

Acknowledgments

This work was supported by the Computational Science Graduate Fellowship Program of the Office of Scientific Computing in the Department of Energy. The National Science Foundation provided initial financial support under grant DDM-8619582.

I give thanks to my advisor, Professor Ross Swaney, for the time we have spent working together on this project and for the help he has given me.

I am grateful to my parents, Bob and Sarah Epperly, for providing me a solid foundation on which to build and for encouraging me to pursue my dreams.

I gratefully acknowledge Jesus Christ for the crucial role he plays in my life and work, and I thank him for the support that parts of his church in Madison have provided to me. Specifically, I would like to recognize Inter-Varsity Christian Fellowship, Faith Community Bible Church (especially the Living Stones house group), my teacher Professor Archie MacKinney, the Wayland community, and East Crossing Church.

I would also like to thank my friend of several years and wife of four months, Meg Epperly, whose love and support have made all of the difference. Meg, you have brought me immeasurable joy.

Contents

Chapter 1

Introduction

Many interesting and useful chemical engineering modeling and design problems can be formulated using a mathematical program, a mathematical formalism for defining optimization problems. Phase and chemical reaction equilibrium problems at constant temperature can be solved by minimizing the Gibbs free energy of the systems. In this case, the objective function is the Gibbs free energy, and the constraints provide the models to calculate the Gibbs free energy for the components [67, 69]. Mathematical programming is also a natural tool in design problems where the problem to be solved can often be stated as what is the best (least cost, maximum production, maximum efficiency, minimum waste) solution to a problem that can be modeled with equalities and inequalities. Some of the mathematical programming models that have appeared in the literature include heat exchanger networks [22, 130], pool blending problems [22], distillation column sequencing [2, 22, 128, 129], complex reactor networks [22, 56], fuel mixtures [91], and process flowsheets [54, 55, 100]. Many more problems can be formulated and solved by design engineers provided that reliable algorithms are available to solve them.

The field of optimization is usually broken into the following categories: integer programming (IP), linear programming (LP), nonlinear programming (NLP), mixed integer linear programming (MILP), and mixed integer nonlinear programming (MINLP). LPs and NLPs are problems involving continuous variables with linear or nonlinear functions. IPs are optimization problems where the variables can only take on discrete, integer values. MILPs and MINLPs are problems with both discrete and continuous variables. This dissertation focuses on the solution of NLPs; however, the algorithm contained herein can be extended to problems with binary integer variables by using continuous variables with nonconvex quadratic constraints to force them to take on integral values.

This dissertation presents a new branch and bound algorithm for solving problems of the following general form:

$$\min \quad f(x) \tag{1}$$

$$\text{s.t.} \quad g(x) \leq 0 \tag{2}$$

$$h(x) = 0 \tag{3}$$

$$\underline{x} \leq x \leq \bar{x} \tag{4}$$

$$x \in R^n \tag{5}$$

where f is a function, $R^n \rightarrow R$, and g and h are vector valued functions, $R^n \rightarrow R^m$ and $R^n \rightarrow R^k$ respectively. The method requires that f, g and h be representable as factorable functions, i.e. sums of quadratic terms and single variable functions [66], and that $||\underline{x}||$ and $||\bar{x}||$ be finite. The feasible region, F, of problem (1) is defined as follows:

$$F = \{x \in R^n \mid g(x) \leq 0, h(x) = 0, \underline{x} \leq x \leq \bar{x}\} \tag{6}$$

A solution to problem (1) means finding a value $x^* \in F$ such that $f(x^*) \leq f(x)$ for all $x \in F$. A local minimum of (1) is defined as a point x' such that $f(x') \leq f(x)$ for all $x \in F \cap N_\epsilon(x')$ for $\epsilon > 0$ where $N_\epsilon(x')$ is an ϵ-neighborhood around x' defined by $N_\epsilon(x') = \{x \mid ||x - x'|| < \epsilon\}$. A local minimum is not considered a solution to (1) until it has been proven to be a global solution. Only the minimization problem will be considered because every maximization problem can be rewritten as a minimization problem (e.g. $\max_{x \in F} f(x) = -\min_{x \in F} -f(x)$).

Much of the research into solving problem (1) has focused on the special case of $f(x)$ and $g(x)$ convex and $h(x)$ affine. Under these conditions, problem (1) is considerably easier, and many efficient and proven algorithms exist to solve it. Under such convexity restrictions, a local minimum of (1) is also a global minimum. The methods developed for convex programs use only local information, function values and first and second partial derivatives. With the information from one or more points, they construct an approximation of the original problem which is used to calculate the next point in the iterations. These methods are guaranteed to converge when the problem is convex; otherwise, they may or may not converge to a local minimum. Sequential quadratic programming (SQP) and the projected Lagrangian method used by MINOS 5.4 [73, 74] are two of the more popular local minimization algorithms. When sufficiently near to a solution, these algorithms converge quadratically and superlinearly to the solution.

When the convexity requirements on f, g and h are removed, the problem becomes much more difficult. In fact, Pardalos and Schnitger [85] have shown that checking local optimality for a feasible point when (1) is an indefinite QP is NP-hard, and Pardalos and Vavasis [86] showed that solving a QP with one negative eigenvalue is NP-hard. Solving the simplest of nonconvex NLPs is NP-hard which indicates that the worst case time of any algorithm is going to increase exponentially with the size of the problem unless P = NP [28]. Some of the practical reasons that (1) may be difficult are multiple local minima, a nonconvex and potentially disconnected feasible region, and a lack of simple criteria for verifying global optimality.

Although solving nonconvex NLPs is a proven difficult problem, this does not mean that efficient algorithms cannot be made to solve some classes of nonconvex NLPs; it just means that the worst case performance is exponential in time. There are several examples of algorithms with exponential worst case behavior that are very efficient for “real” problems: the simplex method for LP, the branch-and-bound algorithm for the knapsack problem [28], and the traveling salesman problem.

Because the general nonconvex NLP problem is considered intractable, the scope of this research project is focused on NLPs taken from chemical engineering design problems; nevertheless, the algorithm has been tested on other problems as well. Design problems have special properties because they model physical systems. For example, all of the variables in a design problem are bounded because of the physical quantity they represent or by practical limitations. For flowsheeting problems, many of the constraints are linear; and most of the nonlinearities come from bilinear terms. The more complicated nonlinearities come from the thermodynamic and reaction calculations.

Current global optimization algorithms can only solve small problems. Quesada and Grossmann [96] have solved a problem with 20 variables with a fractional objective and convex constraints and a problem with 10 variables with a quadratic objective and nonconvex quadratic constraints. Ryoo and Sahinidis [105] report solving a problem of 12 variables with a nonconvex objective and nonconvex quadratic constraints. Phillips and Rosen [87] have solved concave QPs with 50 nonlinear variables and 400 linear variables. Mfayokurera [69] has solved an equilibrium problem with 16 variables, and Swaney [114] has solved problems of 2–8 variables. Horst and Tuy [49] have solved concave minimizations of up to 50 variables. Visweswaran and Floudas [127] have solved indefinite QPs of 20 variables, a quadratic NLP with 39 variables, and randomly generated concave and indefinite QPs of 25 variables.

The algorithm presented in this dissertation is a branch and bound algorithm using rectangular variable partitions. Lower bounds are provided by a new bounding technique which is an extension of Swaney's bounding technique [114] which involves the solution of a LP at each branch and bound iteration. This LP is able to provide the exact lower bound over regions of finite size thus providing for finite rather than ϵ termination of the branch and bound algorithm. Upper bounds are provided by the objective function value of the best feasible point, and the algorithm considers two different strategies for locating improved feasible points. The first uses MINOS 5.4 [74] in the search for local minima, and the second uses a new algorithm using the bounding LP to provide a search direction in an iterative scheme. The bounding LP may provide a descent direction when far away from a minimum or when situated at a nonglobal local minimum. Branching is achieved by choosing a variable and splitting its domain; these choices are made using information from sensitivity analysis of the bounding LP.

The remainder of this report proceeds as follows. In Chapter 2, a summary of the history of the field of global optimization is given highlighting related work. In Chapter 3, a LP underestimator for a particular orthant (one of the 2^n sub-rectangles of the current rectangular region) is developed using McCormick's convex envelopes and new constraints based on positive definite combinations of quadratic terms. In Chapter 4, a method to combine the 2^n orthants into a single underestimating LP is presented. The combination of the orthant programs can be viewed as a LP with interval coefficients, so a method for transforming a LP with interval coefficients into a LP with real coefficients is developed. The interval form of the orthant program is simplified and tightened by eliminating some of the variables, and then it is transformed into a conventional LP with fixed (noninterval) coefficients. This transformation avoids the solution of 2^n subproblems while retaining finite termination characteristics on nonconvex NLPs.

The sequential versions of the algorithm appear in Chapter 5. That chapter contains the details of the branch and bound algorithm, the sensitivity-based branching rules, the region analysis procedure using MINOS 5.4 [74], the region analysis procedure using the underestimating LP, and the problem scaling heuristics. Chapter 6 covers the parallel brand and bound algorithm including the communication protocol used to share work between processors. The results of applying the sequential and parallel algorithms to 50 test problems are given in Chapter 7, and Chapter 8 provides some concluding remarks.

Chapter 2

History of Global Optimization

The study of global optimization is a little more than 30 years old. The first published algorithm for solving a nonconvex NLP is Tuy's [118] paper on solving NLPs with concave objective functions and linear constraints. Now the field has grown to the point that several monographs have been written on the subject [22, 23, 33, 49, 66, 84, 98, 117] and several review articles have appeared [40, 64, 80, 83]. Because the global optimization problem is so difficult, much of the research has focused on solving problems with special properties, for example: concave programs, concave and indefinite QPs, programs of factorable functions, or bilinear programming problems. This summary of the global optimization literature is organized primarily by the type of algorithm being used and occasionally by the type of problem being solved.

In categorizing global optimization algorithms the first main distinction is between stochastic and deterministic methods. Stochastic methods all use a random factor in their algorithm and rely on a statistical argument to prove their convergence. Recent work in this area is summarized in the 1989 monograph [117] and the 1991 review paper [108]. Typically, stochastic methods are applied to unconstrained problems where all the variables have upper and lower bounds. Stochastic methods are good because they do not require any *a priori* assumptions about the objective function, and they often give optimal or near optimal results. The disadvantages of these methods are that they are not guaranteed to find a solution in a finite number of steps, and they cannot realize when they have or have not found the global minimum. For constrained problems they may not even find a feasible point.

Deterministic methods in comparison do not make random choices in their algorithms, and their convergence proofs do not rely on probability. Some offer finite termination for specific problems, and others converge only as the number of iterations approaches infinity. The deterministic methods may be further broken down as cutting plane, branch and bound, interval analysis, generalized Benders decomposition based, and other algorithms.

A recent article by Phillips and Rosen [90] presents a computational comparison of a stochastic and deterministic algorithm applied to the same set of over 140 linearly constrained concave problems, and for this set of problems the deterministic method is always faster.

2.1 Cutting Planes

The strategy of cutting plane algorithms is to locate a local minimum of the problem and to generate a linear constraint that eliminates that local minimum and part of the feasible region near the local minimum from the feasible region. Repeating this procedure eventually eliminates all of the feasible region thus solving the problem. The success of the methods depend on how much of the feasible region each cut can remove; a cut which removes more of the feasible region is called a deeper cut.

The earliest deterministic algorithm for solving a nonconvex NLP is the cutting plane algorithm developed by Tuy [118]. Tuy considers minimization of a concave function over a convex polyhedron. It is well known that the solution of this problem occurs at a vertex of the polyhedron which makes it simpler to consider. Tuy's strategy is to locate a vertex that has an objective value better than or equal to all of its neighboring vertices and to construct a plane that because of the concavity of the objective function removes part of the feasible region around the current point. The algorithm involves solving a sequence of linear programs to determine the solution in a finite number of iterations. Zwart [132] tests Tuy's cutting plane algorithm on a variety of small test problems and reports poor performance. The algorithm fails to solve a program of ten variables because the cutting plane fails to eliminate much of the feasible region.

Ritter [16] proposes a cutting plane algorithm for solving indefinite QPs. He provides a method for detecting local and global minima, a method for finding a local minimum of the QP, and a method for generating a cutting plane to remove a local minimum and other points which cannot contain the minimum from the feasible region. When the feasible region is bounded, the algorithm terminates after a finite number of iterations, but it may take a great deal of computational effort. Problems with several positive and negative eigenvalues also cause the algorithm to be less efficient.

One of the keys to an efficient cutting plane algorithm is generating cuts which eliminate as much of the feasible region as possible without eliminating any points that could contain the solution. Balas and coworkers [6, 7, 9] apply cutting planes to integer programming problems and develop deeper cuts by projecting the rays onto a circumscribing hypersphere [6], a dual convex polyhedron [9], and generalized outer polars [7]. In a later paper, Balas [8] presents a cutting plane algorithm for indefinite QPs using generalized polars. The polar set indicates a region that cannot contain the solution which can be used to generate a cutting plane. He also presents an algorithm for searching vertices outside the eliminated area.

Konno [57] considers the solution of the bilinear program,

$$\begin{aligned} \max \quad & c^{1\mathrm{T}}x^1 + c^{2\mathrm{T}}x^2 + x^{1\mathrm{T}}Cx^2 \\ \text{s.t.} \quad & A^1x^1 = b^1 \\ & x^1 \geq 0 \\ & A^2x^2 = 0 \\ & x^2 \geq 0 \end{aligned}$$

His cutting plane algorithm exploits the special properties of the bilinear programming problem to provide a deeper cut. This cut removes part of the feasible space that cannot contain a solution within ϵ of the current solution. The algorithm proceeds by removing parts of the feasible region until it has all been eliminated and the best point has been found. Konno [58] later extends this work to cover minimization of a concave QP by establishing an equivalent bilinear program and using his previous cutting plane algorithm. The extended work also includes a method for iterative improvement of the cutting planes. By exploiting the special structure of the problem, the method provides cuts typically much deeper than those of Tuy or Ritter. His algorithms are ϵ optimal which means that they return a solution whose objective is guaranteed to be within ϵ of the global optimum.

Vaish and Shetty [123] develop a cutting plane algorithm for the bilinear programming problem which introduces cuts for only a subset of the variables. The assumption of this method is that some of the variables have constraints of a special structure which allow them to be solved more efficiently. This special structure would be destroyed if cutting plane constraints were added, so the cutting planes are limited to a subset of the variables. Their cut is derived using generalized polars to obtain a deeper cut, and their algorithm produces the optimal solution rather than an ϵ optimal solution. However, finite termination has only been proven for ϵ optimality. This algorithm is presumably more efficient for problems with variables and constraints in a special structure. Sherali and Shetty [110] present a finitely convergent cutting plane algorithm using polar cuts and disjunctive face cuts. Similar to previous work, they only introduce cuts in one set of variables to preserve the special structure of the other set's constraints. Disjunctive face cuts remove an extreme facet of the feasible region.

As can be seen from this overview, cutting plane algorithms have been proposed for linearly constrained concave programs, IPs, indefinite QPs, and bilinear programs. Except for the IPs, all of these problems are linearly constrained, a condition too restrictive for engineering design problems. Additional work on cutting planes has occurred in the context of branch and bound algorithms. This work is included in the section below.

2.2 Branch and Bound

Branch and bound algorithms for nonconvex NLPs generate lower and upper bounds for the problem over a particular sub-domain of the original domain. The algorithms converge to a solution by raising the lower bound and reducing the upper bound until they are equal. The methods of obtaining lower bounds vary from algorithm to algorithm, and the upper bound is available from the best known feasible point. Typically, at the outset the lower bound is much lower than the global minimum of the problem, so the algorithm improves the lower bound by splitting the variable domain into two or more subsets, the union of which is the original domain. These subsets, called regions, are stored in a list, and the lower bound for the problem is the least lower bound for all of the regions in the list. As the region's size decreases, the lower bound improves and approaches the global minimum of the problem.

A 1969 paper by Falk and Soland [20] presents the first branch and bound algorithm for solving nonconvex NLPs. Their algorithm is applicable to problems with a separable objective function (the sum of single variable functions) and a constrained, convex feasible region. It requires lower and upper bounds on each variable which define a n-dimensional rectangle, and uses rectangular partitions to subdivide the original domain into regions. To provide lower bounds over a rectangular region, the algorithm uses a convex underestimating NLP. This NLP is derived by replacing the original objective with its convex envelope. The convex envelope of a function over a set C is the pointwise maximum of all convex functions which underestimate the function for each point in C. The underestimating NLP is convex, and its minimum represents a lower bound for the original problem over the variable sub-domain. The variable that the algorithm chooses to split is the variable with the greatest difference between the function which appears in the objective for this variable and its convex envelope. In this way, it attempts to improve the fit of the convex envelopes at each branch and bound iteration.

Soland [113] extends Falk and Soland's work to treat problems with separable nonconvex constraints by replacing the constraint functions with their convex envelopes. This produces an algorithm with proven yet perhaps infinite convergence. Another early branch and bound algorithm is the one developed by Beale and Tomlin [11] which uses the concept of ordered sets of variables. Their method uses piecewise linear approximations, so the subproblem to be solved is a LP. McCormick [64] provided a summary of the early work in global optimization and recommended pursuing the branch and bound methods.

One of the chief limitations of Falk and Soland's algorithm and Soland's extension are that they can only be applied to separable functions. McCormick [65] removes this limitation by developing rules for generating convex/concave bounding functions for factorable functions, which include the functions occurring in most common NLPs. This extension makes it possible to apply branch and bound to nearly all NLPs. In the second part of his paper, he develops a splitting rule which is a generalization of the one used by Falk and Soland. It is based on splitting the variable appearing in a function whose envelope differs maximally from the true function value at the solution of the bounding program. This splitting rule attempts to improve the fit of the bounding functions. The algorithm presented in this dissertation incorporates McCormick's convex/concave envelopes for bilinearities in its bounding strategy.

For the remainder of the chapter, "general algorithm" is used to refer to an algorithm type that covers a wide class of algorithms and requires further specifications before it can be implemented, and "algorithm" or "concrete algorithm" refers to an implementable algorithm. General algorithms exist primarily to provide proofs for large classes of algorithms, and concrete algorithms exist to solve problems.

Horst [36] presents a general branch and bound algorithm to remove the dependence on convex envelopes and rectangular partitions in the convergence proofs of previous work. In their place, he uses general underestimating functions and compact partitions, and provides criteria that these unspecified pieces must satisfy to prove convergence. He provides a formal definition of a partition: a combination of sets K^i are a partition of set K if and only if $\cup_i K^i = K$ and $K^i \cap K^j$ lies on their boundaries. He provides a convergence proof for the general branch and bound algorithm and presents a concrete algorithm for solving problems with a concave objective and a compact, convex feasible region using n-dimensional simplices (n-simplices) to partition the domain. He uses the convex envelope of the objective in his underestimating program. Each simplex is defined by the convex hull of $n+1$ vertices, and the convex envelope of a concave function over a simplex is just a linear interpolation of the values of the function at each vertex. Simplices are more efficient for concave functions because they only have $n + 1$ vertices as compared to 2^n for rectangular partitions. In his initial paper, Horst proves that one accumulation point of the sequence of points generated is the global solution, and later he [37] shows that every accumulation point solves the problem.

Falk and Hoffman [19] propose an algorithm for minimizing a concave function in a polyhedral set. They use an outer approximation approach similar to the methods suggested by Cheney and Goldstein [14] and Kelley [53] for solving convex problems. The method uses an enclosing simplex that is successively improved by adding new vertices until the simplex is close enough to the feasible region to verify the global optimum. The algorithm makes use of the well known fact that the minimum of a concave function over a simplex occurs at a vertex. The algorithm is finite and it involves the solution of a LP at each iteration. One difficulty of this method is that the problem size grows each iteration. Hoffman [35] extends this work to treat more general compact convex feasible regions. Under these conditions, the algorithm converges in the infinite limit. One of the challenging problems for these methods is locating the new vertices after adding a new plane to the polyhedral set. Horst, de Vries and Thoai [41] present an improved method for finding new vertices in outer approximation algorithms like that of Falk and Hoffman, and they also provide methods of identifying redundant constraints.

Thoai and Tuy [116] develop a concrete branch and bound algorithm using conical partitions, and they establish the consistency and exhaustive criteria for proven convergence. The problems they address have a concave objective and

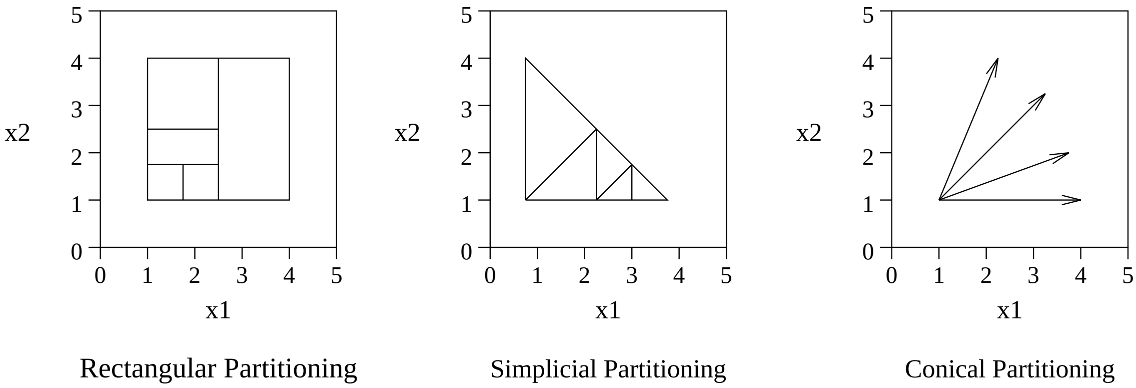


Figure 1: Rectangular, simplicial, and conical partition sets

a polyhedral constraint set. An LP based on Tuy's cutting planes [118] is used to obtain lower bounds for each conical region.

At this point in this chapter, the three main types of partitions used in branch and bound have been introduced: rectangular, simplicial, and conical; these partitions are illustrated in Figure 1. Rectangular partitions are typically used by methods that employ convex envelopes or interval analysis, and simplicial and conical partitions are used for concave problems and other bounding methods.

Benson [12] presents a more general prototype branch and bound algorithm with the introduction of an ϵ convergence tolerance and fewer requirements on the underestimating functions. Falk and Soland's [20] and Horst's [36] algorithms can be seen as special cases of Benson's presentation. Benson is able to prove that any accumulation point of the general prototype branch and bound algorithm is a solution, thereby proving the result for the previous work as well. McCormick [66] gives a detailed explanation of factorable functions and his method for determining the convex/concave envelope of a factorable function. He also includes a concrete branch and bound algorithm using a combination of a local NLP algorithm to find good local minima and a convex underestimating program using the envelopes he developed to attempt to verify the minimum. Calculating a lower bound involves solving a convex NLP. Evaluation of this work is difficult in that no detailed computational results have been reported in the literature.

Al-Khayyal and Falk [4] look at problems with a bilinear objective function and a compact, convex feasible region. Convex envelopes identical to those of McCormick [65] are applied to the bilinear terms in a branch and bound algorithm. They use rectangular partitions, and their splitting strategy is based on the maximal differences between the actual value of a bilinearity and its envelope's value at the solution of the bounding program. Their algorithm splits two variables per iteration — one for each part of the bilinear term — thus producing four regions. When the constraint set is linear, the subproblem is a LP; otherwise, it is a convex NLP.

Rosen [101] introduces a two stage algorithm for minimizing a concave function over a linearly constrained feasible region. In the first stage, the global maximum is determined using a local method. The eigenvectors of the Hessian at the maximal point are used as a heuristic to find vertices of the feasible region "far" from the maximum. These vertices are located using a multiple-cost-row LP. The vertex with the minimal objective value provides an upper bound on the solution and is used to generate a rectangular set in which the minimum may not occur due to the concavity of the objective. Excluding this rectangle from the feasible region leaves at most $2n$ polyhedral subdomains which are solved in the second stage using the Falk-Hoffman [19] algorithm, which is expected to be efficient because "tight" initial simplices can be provided. Rosen [102] analyzes the performance of several approximate branch and bound algorithms and provides a bound on the computational effort required for an algorithm.

Tuy, Thieu and Thai [121] relax the requirements of the Thoai-Tuy algorithm [116] that required a bounded polyhedral feasible region. They develop a conical branch and bound algorithm for concave objectives over a closed, possibly unbounded convex set. They consider a number of different splitting strategies: splitting the oldest cone, splitting the largest cone, and splitting the most promising cone. To obtain lower bounds over a cone, the cone is cut by a supporting hyperplane of the convex set.

Rosen and Pardalos [103] study QPs with relatively few variables appearing in the concave part of the objective and more variables appearing linearly. The problem is converted to a separable problem by solving a multiple-cost-row LP with $2n$ cost rows. They approximate the original problem with piecewise linear functions designed to give the optimal value with ϵ tolerance which is solved using a MILP. This MILP may be difficult to solve because it may have numerous integer variables although they have a special structure.

Horst [38] further generalizes the Benson-Falk-Horst-Soland (BFHS) algorithm, the name given to the algorithm presented by Benson [12] which includes work by those named, to include the branch and bound algorithm of Tuy, Thieu, and Thai [121] which uses conical partitions and a cutting plane approach to obtain lower bounds. He makes other improvements to allow region deletion and different variable selection rules. He derives the convergence criteria for the general algorithm and proves the criteria for simplicial, conical and rectangular partitioning strategies. The general algorithm is proven to converge in the infinite limit. Horst includes a concrete algorithm for minimizing concave functions over compact, convex sets.

Pardalos, Glick and Rosen [81] extend the work of Rosen and Pardalos [103] to treat indefinite QPs. Indefinite QPs may not have a vertex solution, so they are more difficult in some senses than concave QPs. Pardalos, Glick and Rosen transform the problem objective into a separable quadratic function using the eigenvalues, eigenvectors, and a multiple-cost-row LP. The objective function can be viewed as the difference of two convex functions. The concave part is underestimated with its convex envelope (a linear interpolation of the endpoints), and a limited number of branch and bound iterations are applied. Lastly, the authors use a piecewise linear approximation of the objective function [103] to build a MILP which provides an ϵ optimal solution. They report solving problems of up to 30 concave variables combined with 300 convex variables.

Horst and Tuy [48] show how a variety of other algorithms can be subsumed under the general branch and bound algorithm. Specifically, they address Pintér's general class of algorithms [93, 94], the approach of Zheng and Galperin [26, 27, 131] and the algorithms of Pijavskii [92], Shubert [112] and Mladineo [70]. In a subsequent paper, Tuy and Horst [120] review the convergence criteria for the general branch and bound algorithm. They introduce a restart branch and bound algorithm for use in outer approximation algorithms where the feasible set is overestimated with linear cuts. Their method incorporates new cuts for the feasible region as it progresses. They include concrete algorithms for concave minimization and programs that can be expressed with functions that are the difference of two convex functions, called d.c. programs. D.c. programs can be converted to a program with a linear objective, one reverse convex constraint, and one or more convex constraints. A reverse convex constraint is one of the form $g(x) \geq 0$ with

$g(x)$ convex. Both concrete algorithms are finite.

With the exception of Falk and Soland's algorithms [20, 113] and McCormick's algorithm [65, 66], most of the algorithms presented to this point involve problems that are too limited for most engineering applications. However, many programs can be expressed as d.c. programs. "The intuition of many people working in global optimization is that most of the 'reasonable' optimization problems actually involve d.c. functions [49]." It may be non-trivial to find the d.c. form for a particular problem; nevertheless, d.c. programming algorithms have been applied to some chemical engineering programs. McDonald and Floudas [67] show that an equilibrium problem using the UNIQUAC equation is a d.c. program and solve it using a branch and bound algorithm.

Phillips and Rosen [87] present a parallel branch and bound algorithm for solving concave QPs with many variables appearing linearly. The concave part is in separable form and is bound by the convex envelope. They use a two stage algorithm. In stage one, a multiple-cost-row LP is solved to produce a lower bound and to predict parts of the region that can be eliminated. A multiple-cost-row LP is used because the algorithm considers bisecting each quadratic term individually to produce tighter bounds and to predict a subset of the feasible region that can be eliminated. The second stage is a branch and bound algorithm. Parallelism is achieved through the parallel solution of the multiple-cost-row LP. They solve problems of 50 nonlinear variables and up to 400 linear variables on a four processor CRAY 2. They later extend this work to treat indefinite QPs [88]. They observe good and sometimes superlinear speedup.

One of the assumptions in the general branch and bound algorithm developed by Horst, Benson and Tuy [12, 38, 120] is that feasible points can be easily found inside of each partition. For simple convex feasible regions, this may be the case, but finding a feasible point can be a considerable task in itself. Horst [39] extends his general branch and bound framework and convergence proofs to situations where the feasibility of the partition sets is not known. Such is the situation for the algorithm presented in this dissertation; feasible points are not readily available in each region being searched. In addition, Horst and Thoai [42] make use of this refinement to the algorithm while applying global optimization to solve a system of Lipschitzian equations and inequalities.

Al-Khayyal and Larsen [5] apply branch and bound using convex envelopes to optimization of indefinite quadratic objective functions subject to a linearly bounded mixed integer constraint set. Their approach can be used for pure integer and pure real problems as well.

One of the difficulties with many branch and bound algorithms is that they are only convergent in the infinite limit or finitely to ϵ optimal solutions. This problem is addressed by Swaney [114]. He develops a new bounding program which employs a combination of McCormick's convex envelopes [65] and new constraints based on positive semi-definite combinations of quadratic terms. This new bounding problem is able to produce the exact lower bound over a region of finite, nonzero size when constructed at the region's global minimum. This ability gives the branch and bound algorithm finite termination. Swaney also introduces a splitting strategy based on a sensitivity analysis of the bounding program. This strategy chooses the variable which is predicted to raise the lower bound the most and a split location which will often result in one of the two regions being pruned.

The algorithm presented here is a direct extension of this work.

Horst, Thoai, and Benson [45] present a more efficient branch and bound algorithm for concave minimization over a convex feasible region. They use a conical partition and a polyhedral outer approximation of the convex feasible region. This makes the lower bound step a LP instead of a problem with a linear objective and convex constraints, resulting in improved efficiency.

Many branch and bound algorithms use bisection to achieve an exhaustive subdivision process which is required for proven convergence; however, convergence of algorithms based on bisection can be slow. Tuy [119] develops less restrictive requirements for the subdivision strategy to enable a wider class of splitting rules to be employed in algorithms with proven convergence.

Simplicial partitions are used in several branch and bound algorithms, but radial subdivision is limited because it cannot be related to the behavior of the objective function, and it does not adapt to the extreme point optimality of concave programming problems. A new type of splitting strategy is introduced by Horst, Thoai, and de Vries [47]. Simplicial covers are constructed with the objective function in mind, so that large parts of the feasible region can be eliminated at each iteration. The primary difference between simplicial cover and partitions is that covers can overlap each other by more than just their boundaries. Horst, Thoai, and de Vries [46] construct a branch and bound algorithm using simplicial covers instead of partitions. They use outer approximations for the constraint set and incorporate some cutting planes to remove part of the interior of the feasible region. This algorithm has proven superior to the previous work of Horst, Thoai, and Benson [45] for 26 test problems.

Horst and Thoai [43] present an algorithm for concave problems with numerous variables appearing linearly. This problem is similar to the ones treated by Phillips and Rosen [87] and Rosen and Pardalos [103]. This approach uses a conical branch and bound algorithm with dimensions only one greater than the number of nonlinear variables, and it requires only one LP subproblem at each iteration.

Al-Khayyal, Horst and Pardalos [3] consider global optimization problems arising in nonlinear bilevel programming. These give rise to minimization of concave functions subject to quadratic constraints in separable form. They propose a branch and bound algorithm using convex envelopes and an approach based on a piecewise linear approximation using 0-1 variables. Preliminary results favor the branch and bound method.

Obtaining tight lower bounds for the NLP is the key to the success of branch and bound algorithms. Sherali and Alameddine [109] develop a new bound technique for bounding bilinear programming problems which they call the reformulation-linearization-technique (RLT). They reformulate the problem by multiplying linear constraints to form implied constraints on the bilinear terms. These constraints include the convex envelopes of McCormick [65] and are shown to be tighter in some cases. After multiplying the constraints, all of the bilinear terms are linearized by replacing each bilinear term with a variable. This produces a set of linear constraints that produce tighter bounds on the bilinear terms. They apply RLT in a branch and bound algorithm. The difficulty with this method is that the LP size grows exponentially with the number of constraints. If supplemented by a method to determine which constraints are necessary, this technique would be much more useful. Sherali and Tuncbilek [111] extend this work by treating polynomial NLPs.

The approach is basically the same. Extra constraints are generated from products of the variable bound constraints and also from the general constraints as an option. Again, it produces bounds tighter than McCormick's envelopes at the cost of exponential growth in the LP size.

Muu [75] presents a branch and bound algorithm for problems with convex objective functions and convex constraints with an additional convex-concave constraint. He uses a simplicial partition with adaptive bisection. Lower bounds are provided with a relaxation of the original problem. Muu and Oettli [75] present a general combined branch and bound and outer approximation algorithm. Concrete applications are provided for minimizing a convex-concave function over a convex feasible region and minimizing convex functions over a convex constraint set with an additional convex-concave constraint.

One of the tacit assumptions of many global optimization algorithms is that lower and upper bounds will be available for each of the variables. Hamed and McCormick [32] look into the problem of providing bounds on problem variables in a nonconvex NLP. They present a number of different approaches to providing bounds depending on the type of NLP.

Benson and Sayin [13] add a new neighbor generation process to produce a finite branch and bound algorithm for concave minimization over a compact polyhedron. The advantages of their algorithm are that it requires neither nonlinear computations nor determination of convex envelopes nor underestimating functions, and it only solves LPs which differ from each other in one column of the constraint matrix. Their domain partitioning strategy is simplicial.

Horst and Thoai [44] provide a branch and bound algorithm for the canonical d.c. programming problem. This problem involves a linear objective function and a feasible region which is the intersection of a convex set and the complement of an open convex set (a reverse convex constraint). They use a decomposition strategy to solve a problem in n dimensions where n is the dimension of the reverse convex constraints. They develop two algorithms to solve this problem: the first employs a conical branch and bound algorithm using outer approximations, and the second uses a cutting plane algorithm.

Ryoo and Sahinidis [105] provide an algorithm with new methods for reducing the variable domain based on optimality and feasibility arguments. They incorporate these variable domain reduction techniques into a branch and reduce algorithm based on rectangular partitions and convex envelopes to provide lower bounds. Their variable reduction techniques are a useful addition to the algorithm because they give a computationally cheap method of reducing the variable domain. They report solving engineering oriented problems with 2–12 variables.

Quesada and Grossmann [96] present a generalization of the algorithm they developed for heat exchanger networks [95]. Their algorithm focuses on fractional and bilinear NLPs. They develop a branch and bound algorithm using McCormick's convex/concave envelopes [65], envelopes for fractional terms, and additional estimators from the projection of hyperplanes at the boundary of the feasible region. They consider bounding bilinear terms as either $q_{ij} = x_i x_j$ or as $x_i = \frac{q_{ij}}{x_j}$, and they provide rules to detect which estimators are nonredundant. For bilinear terms, the additional constraints are equivalent to a limited number of the constraints proposed by Sherali and Alameddine [109].

Many of these branch and bound algorithms are too restrictive in the problems they solve to be useful in solving engineering design problems where the constraints are nonlinear and nonconvex. The exceptions tend to follow the approach of using convex envelopes starting with Falk and Soland [20]. Soland [113] provides a method capable of treating separable nonconvex feasible regions, and McCormick [65] extends it to the whole class of factorable functions. Algorithms for solving d.c. programs [44, 49, 67, 120] have enough scope to treat some engineering problems, but the problems would have to be transformed to d.c. form. Swaney [114] provides a new bounding algorithm to treat algebraic NLPs with an extension for transcendental functions. Sherali and Tuncbilek [111] present an algorithm with considerable scope, but it is limited by the size of the LP it requires. Ryoo and Sahinidis [105] and Quesada and Grossmann [96] both provide algorithms making use of convex/concave envelopes. The algorithm presented here also relies heavily on the concepts of convex/concave envelopes and the additional constraints provided by Swaney [114].

2.3 Interval Methods

The previous section on branch and bound algorithms left out a major class of algorithms: those based on interval analysis. Much of the work on interval analysis takes place independently from the work on branch and bound presented above, so these methods will be presented as a special subset of branch and bound algorithms. Because the algorithm presented here cannot truly be considered an interval method, only a brief overview of recent interval methods will be included.

Neumaier's book [76] provides an excellent introduction to interval analysis in the context of solving systems of equations, and books by Hansen [33] and Ratschek and Rokne [98] cover many of the available techniques for global optimization.

The primary virtues of interval methods are their wide applicability, guaranteed convergence and explicit treatment of machine rounding difficulties. Interval methods can be applied to discontinuous or nondifferentiable functions; however, their performance can be improved with acceleration devices if the functions are differentiable. Interval methods have proven convergence under very general circumstances. By representing numbers as intervals, they provide a method for treating machine rounding problems explicitly rather than just trusting that they are insignificant. However, in some cases interval algorithms can be very slow.

Interval analysis is an extension of real arithmetic to intervals with upper and lower bounds. An interval x can be defined as follows:

$$x = [x^L, x^U] = \{x \in R \mid x \geq x^L, x \leq x^U\}$$

Given this initial definition, one can provide methods for evaluating all of the common arithmetic operations, $+$, $-$, $\times$, $\div$ [71].

$$[a,b] + [c,d] = [a+c, b+d] \tag{7}$$

$$[a,b] - [c,d] = [a-d, b-c] \tag{8}$$

$$[a,b] \times [c,d] = [\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}] \tag{9}$$

$$[a,b] \div [c,d] = [a,b] \times [1/d, 1/c] \text{ if } 0 < c \text{ or } d < 0. \tag{10}$$

Interval analysis explicitly treats machine rounding problems by outward rounding each result to guarantee that the range of a function is included in the interval result. Ranges for more complicated functions can be determined by replacing normal real arithmetic with interval arithmetic and developing interval extensions for any transcendental functions needed. Replacing real operations with interval operations gives what is called the natural interval extension of a function — more precise methods of bounding functions are presented in [76]. These methods provide an interval guaranteed to include the true range, but it may overestimate the range considerably on large intervals.

Ratschek [97] presents a basic interval algorithm for solving bound constrained global minimizations, problems with finite upper and lower bounds for each variable and no other constraints. The technique requires an inclusion function $F(Y)$ for the objective function $f(y)$ where Y is an interval and $f(y) \in F(Y)$ for all values of $x \in Y$. The algorithm keeps a list of intervals and the lower bound for each interval evaluated by $\min F(Y)$. The list is sorted so that the lower bounds always increase, which causes the least lower bound to be at the front of the list. At each iteration, the front element of the list is removed, and its widest edge is bisected. The resulting two intervals are bounded with $F(Y)$, and the results are added to the list in the correct order. Iterations continue until the width of the enclosure of $F(Y)$ decreases to a preset tolerance ϵ. The lower bound of the first element in the list provides the solution to the problem. Centered forms (e.g. the mean-value form or the Taylor-forms) can be used to improve the convergence rate of the algorithm.

Moore and Ratschek [72] present two improvements to this basic algorithm. The first addition reduces the list size through the "midpoint-test." It discards all elements of the list whose lower bound is greater than $F(c)$ where $c = \text{mid } Y$ or any other point in Y. The second modification is to enter elements in the list so the widths of the intervals do not increase and to apply the midpoint-test to the interval with the minimal midpoint.

Ratschek and Voller [99] provide an overview of interval tools available for global optimization algorithms. They present the following additional tests for eliminating intervals from the list: the monotonicity test, the nonconvexity test and interval Newton operators. If an inclusion of the gradient of the objective indicates that the objective is monotonic over an interval, then only one of the end points need be considered: this is the monotonicity test. The nonconvexity test involves testing the bounds on the Hessian of the objective. The interval Newton methods are the interval equivalent of solving $\nabla f(x) = 0$ using interval matrix techniques. These can reduce the width of the intervals that could contain the solution. Ratschek and Voller also describe how nonsmooth and constrained optimization can be treated.

Kearfott [51] proposes interval Newton methods for solving $\min f(x)$ over a bounded set by solving $f'(x) = 0$ by interval analysis. He considers the case when the solution to $f'(x) = 0$ is near a singularity. He uses a preconditioned interval Gauss-Seidel method to reduce the range of the interval bounds on x. In a more recent work [52], he presents an interval-based branch and bound algorithm for differentiable nonlinear functions over an interval. He improves the behavior of the Gauss-Seidel variable reductions by using optimal preconditioners determined by an LP.

Recent work by Vaidyanathan and El-Halwagi [122] presents an interval based branch and bound algorithm with applications in chemical engineering. To eliminate regions they propose the standard deletion rules: an upper-bound test (similar to the midpoint-test), the infeasibility test, the monotonicity test , and the nonconvexity test. In addition to the normal deletion rules, Vaidyanathan and El-Halwagi introduce a lower bound test and a distrust-region method. Using another method such as McCormick's bounding program [65], they obtain a valid lower bound on the solution. Any region whose upper bound on the objective is less than the lower bound obtained by the other method must be infeasible and can be deleted. When an infeasible point is discovered, the distrust-region method attempts to remove the largest box of infeasible points around that point. In addition, GINO is used to find local minima to accelerate the solution.

2.4 Generalized Benders Decomposition

Generalized Benders decomposition (GBD) [29] is a technique for solving problems with the variables divided into two sets, x and y. The y variables are called complicating variables, and it is assumed that the problem in x becomes much easier to solve when y is fixed. GBD transforms the problem into an embedded optimization problem which is converted into a master problem in the y variables and a subproblem in the x variables. It solves the original problem through a sequence of these two problems.

The first attempt to apply GBD to global optimization is the work by Kough [60], which applies GBD to indefinite QPs that result in a sequence of convex and concave problems. To solve the concave subproblem, Kough uses a cutting plane method developed by Cabot. Although finite, this must be a computationally intensive algorithm because it involves such a difficult subproblem.

Floudas, Aggarwal and Ciric [1, 21] apply GBD in a local algorithm for solving QP, NLP, and MINLP problems; however, their algorithm is not guaranteed to converge to a solution. Although the algorithm frequently converges to the solution, Floudas and Visweswaran [24] report that it fails in certain cases. Sahinidis and Grossmann [106] carefully address the topic of the convergence of GBD when applied to nonconvex NLPs, and they show that it will not necessarily converge to the global minimum from all starting points. In fact, they consider an example in which it will rarely converge to the global minimum. They show that if the original program has q local minima, the projected problem will have q local minima too.

However, later Floudas and Visweswaran [24] address these weaknesses in the original algorithm and develop a global optimization algorithm (GOP). They consider a problem of the following form:

$$\begin{aligned} \min \quad & f(x, y) \\ \text{s.t.} \quad & g(x, y) \leq 0 \\ & h(x, y) = 0 \\ & x \in X \\ & y \in Y \end{aligned}$$

GOP can be applied to problems when: $f(x, y)$ and $g(x, y)$ are convex in x for fixed y and convex in y for fixed x, and $h(x, y)$ is affine in x for fixed y and affine in y for fixed x. Bilinearities are treated by putting one variable in x and the other in y, and polynomials can be converted into bilinearities by introducing additional variables. In this way, GOP can be applied to a wide class of problems.

GOP involves the solution of a sequence of primal and relaxed dual problems. The primal problem is solved for fixed values of y, and during each iteration a number of relaxed dual problems are solved in the y variables, each with different constraint based on a particular combination of bounds for the complicating variables. The number of relaxed dual problems required is 2^q where q is the number of complicating variables. In order to reduce the number of relaxed dual problems required, several heuristics have been implemented [125, 127] which provided orders of magnitude improvement.

Visweswaran and Floudas [124] apply GOP to a variety of chemical engineering and test problems to demonstrate its usefulness. Visweswaran and Floudas [126] show how GOP can be improved for the solution of polynomials in one variable because of the simple form of the problem. Hansen, Jaumard and Xiong [34] demonstrate that this approach is equivalent to an interval algorithm using the natural interval extension to the cord-slope form of Taylor's expansion, and they explore other more efficient interval methods. Liu and Floudas [62] show how GOP can be applied to problems with an objective function and constraints that have continuous second derivatives. This extension is realized through a transformation of variables to give the problem the required properties. Liu and Floudas [63] address the convergence of GOP and show that it terminates after a finite number of steps for a wide class of problems.

2.5 Other Methods

There are a number of other methods that do not fit into the above categories. This is not intended to be an exhaustive list of methods that have been proposed, but rather to present a few of the recent algorithms in the literature.

Thạch and Tuy [115] develop an algorithm to find the minimum of a continuous and not necessarily smooth function over a compact set. Their method is based on the sequential solution of a linearly constrained concave program to either verify a global solution or provide an improved point. This concave program is derived from a relief indicator function that indicates when a feasible point is better than the current local minimum. They solve a sequence of these programs until the global solution is verified. The algorithm may be of use in solving small problems.

Konno and Kuno [59] consider the solution of linear multiplicative and fractional programs. These problems involve an objective function that is either the product or ratio of two linear functions in x. They introduce a parameter into the original problem and consider solutions of the original problem as a function of the parameter. The solution is located by minimizing over the parameter; however, this may be difficult because the function over the parameter has multiple location minima. Pardalos and Phillips [82] also consider a parametric approach to the optimization of fractional programs. They present an improvement of Dinkelbach's algorithm [18] which provides upper bounds and error estimates at each iteration. Except under special cases, the algorithm involves global solutions of nonconvex subproblems at each iteration.

2.6 Summary

Although global optimization has been a topic of research for over 30 years, relatively few practical algorithms exist with enough generality to be applied to chemical engineering problems. The algorithms that can be applied to chemical engineering problems fall into the following categories: branch and bound using convex envelopes and additional bounding functions, branch and bound for d.c. programming problems, branch and bound using interval methods, and GOP (GBD). Of these methods, the work presented here extends the approach based on branch and bound using convex envelopes and additional bounding functions.

Chapter 3

The Orthant Program

Applying a branch and bound algorithm to a NLP requires a technique to calculate a lower bound on the solution to the NLP over a subset of the variable range. For the algorithm to be effective, the technique must provide a lower bound which is as close to the real solution as possible while remaining quickly and reliably solvable. If the gap between the lower bound and the real solution is large, the branch and bound algorithm will have to evaluate more regions to obtain adequate bounds for the solution. If the lower bound cannot be reliably obtained, the algorithm may not terminate.

There is also an unquantifiable trade off between the quality of the bound and the time it takes to determine. A high quality bound will take longer to calculate per region and take presumably fewer regions to solve than a quicker, low quality bound. It is not clear what the optimum trade off is, but this work is focused on obtaining the best possible bound with a reasonably sized LP.

In this chapter, a LP to calculate a lower bound on a NLP for a particular orthant of the rectangular variable domain will be developed and explained. Given a point in the interior of a rectangular variable subdomain, the region is hypothetically split into 2^n n-dimensional rectangles corresponding to the orthants formed by a set of rectangular axes centered at the point. The set of these 2^n orthant LPs could be used to obtain a bound over a subdomain, but this is clearly impractical for large problems. The orthant programs are mainly useful as a step in developing the covering program presented in the following chapter.

The procedure to develop the orthant program from the NLP is summarized as follows. The first step is to transform the original problem into a quadratic NLP, a NLP with a quadratic objective and quadratic constraints. Next, using a variable transformation, the quadratic terms are separated into two groups — those which occur in the range space of the constraints and those which occur in the null space of the constraints. Linear bounds for the constraint space quadratic terms are constructed using McCormick's convex envelopes for a bilinearity, while linear bounds for the null space quadratic terms are constructed from positive definite combinations. Both types of constraints are combined into a LP which underestimates the original NLP for a particular orthant. The details of each step are presented below.

3.1 Transformation to Quadratic NLP

It is assumed that the problem is well scaled and written in the following factorable form:

$$\min \quad x_0 \tag{11}$$

$$\text{s.t.}\quad g_i(x) = b^{i\mathrm{T}}x + x^{\mathrm{T}}H^{(i)}x + \sum_{j=F_i} f_j^i(x_j) \leq 0 \quad i \in 1 \ldots m \tag{12}$$

$$\underline{x} \leq x \leq \bar{x} \tag{13}$$

$$x \in R^n \tag{14}$$

McCormick [66] shows how many problems can be written in this form. Equality constraints are written as two opposing inequality constraints. $H^{(i)}$ is chosen to be a symmetric matrix, and the functions $f_j^i(x_j)$ are nonlinear, single variable functions. For this algorithm, $f_j^i(x_j)$ must be twice differentiable and able to be underestimated over a specified range by a quadratic function. This problem form allows for a very wide class of problems to be considered.

In addition to the problem definition, the transformation needs a specific region defined by $[x^L, x^U] \subseteq [\underline{x}, \bar{x}]$, a point $\check{x} \in [x^L, x^U]$, an estimate, $\bar{u}$, of the Lagrange multipliers at $\check{x}$ and an estimate of the active set, I_A and I'_A.

$$I'_A = \{i \in 1 \ldots n \,|\, \check{x}_i = x_i^U, \bar{u}_i > 0 \vee \check{x}_i = x_i^L, \bar{u}_i < 0\} \tag{15}$$

$$I_A = \{i \in 1 \ldots m \,|\, g_i(\check{x}) = 0, \bar{u}_{i+n} > 0\} \tag{16}$$

These definitions assume that the first n components of the Lagrange vector correspond to the variable bounds followed by m components corresponding to the general constraints. Swaney's algorithm [114] required that $\check{x}$ be a local minimum of (11) and $\bar{u}$ and I_A be the corresponding Lagrange multipliers and active set. The orthant program presented here is more general because it can be applied at any point; when the $\check{x}$ is a local minimum, it is equivalent to Swaney's orthant program.

The first step in the transformation is to replace the single variable, nonlinear functions with underestimating quadratics. The Taylor series expansion of $f_j^i(x_j)$

about $\check{x}$ can be written as

$$f_j^i(x_j) = f_j^i(\check{x}_j) + \left.\frac{df_j^i}{dx_j}\right|_{\check{x}_j}(x_j - \check{x}_j) + \frac{1}{2}\left.\frac{d^2 f_j^i}{d{x_j}^2}\right|_{\check{x}_j}(x_j - \check{x}_j)^2 + \zeta_j^i(x_j) \tag{17}$$

where $\zeta_j^i(x_j)$ holds the higher order terms. Given the expansion, g_i can be rewritten exactly

$$\begin{aligned} g_i(x) &= \overbrace{g_i(\check{x})}^{\bar{g}_i} + \overbrace{\left(b^i + 2H^{(i)}\check{x} + \sum_j \left.\frac{df_j^i}{dx_j}\right|_{\check{x}_j} e^j\right)^{\mathrm{T}}}^{\bar{b}^{i\mathrm{T}}}(x - \check{x}) \qquad (18) \\ &\quad + (x-\check{x})^{\mathrm{T}} \overbrace{\left(H^{(i)} + \frac{1}{2}\sum_j \left.\frac{\partial^2 f_j^i}{\partial x^2}\right|_{\check{x}}\right)}^{\bar{H}^{(i)}}(x - \check{x}) + \sum_{j\in F_i} \zeta_j^i(x_j) \\ &= \bar{g}_i + \bar{b}^{i\mathrm{T}}(x-\check{x}) + (x-\check{x})^{\mathrm{T}}\bar{H}^{(i)}(x-\check{x}) + \sum_{j\in F_i}\zeta_j^i(x_j) \qquad (19) \end{aligned}$$

where e^j is the j'th column of a $n \times n$ identity matrix. Because the function f_j^i are single variables functions, the matrix $\frac{1}{2}\left.\frac{\partial^2 f_j^i}{\partial x^2}\right|_{\check{x}}$ is diagonal. A quadratic underestimator for $g(x)$ can be constructed by replacing the functions ζ_j^i with quadratic underestimators. This is accomplished by replacing the second order and all higher order terms with the tightest quadratic underestimation

$$f_j^i(x_j) \geq f_j^i(\check{x}_j) + \left.\frac{df_j^i}{dx_j}\right|_{\check{x}_j}(x_j - \check{x}_j) + \hat{\alpha}_j^i(x_j - \check{x}_j)^2 \tag{20}$$

where $\hat{\alpha}_j^i$ is the largest value that underestimates f_j^i over the range $[x_j^L, x_j^U]$. With the following definition,

$$\hat{H}^{(i)} = H^{(i)} + \mathrm{diag}(\hat{\alpha}^i)$$

a quadratic underestimation of $g(x)$ can be written as

$$g_i(x) \geq \bar{g}_i + \bar{b}^{i\mathrm{T}}(x - \check{x}) + (x-\check{x})^{\mathrm{T}}\hat{H}^{(i)}(x - \check{x}) \tag{21}$$

3.2 Variable Transformation

Now the original nonlinear program has been transformed into a quadratic NLP, and the next task is to provide bounding functions for the quadratic terms. McCormick [66, pages 387–416] developed the convex/concave envelopes for quadratic terms, and these can be used to provide bounding functions. However in this context, these bounds are insufficient for two reasons explained below. These shortcomings may be overcome by separating the bilinearities into two groups of components, those whose gradients lie in the space spanned by the gradients of the set of active constraints, and those whose gradients lie in the null space. This separation is accomplished through a variable transformation.

The first problem with McCormick's envelopes is that the underestimators match the actual functions only on the boundaries of the region, and for the underestimating program to remain stationary when $\check{x}$ is a global minimum, there must be no error in the underestimation at that point. This point is demonstrated by Figures 2, 3.a, and 3.b which show a bilinear function, its convex envelope, and the estimation error as a function of position. Later in the text, w and d will be used as deviations from $\check{x}$, so the error at $(0, 0)$ prevents this bound from being tight at $\check{x}$. This difficulty is addressed below in section 3.3 by bounding each orthant separately using piecewise convex envelopes.

Concave terms in the Lagrangian give rise to the other problem with convex envelopes. If piecewise convex envelopes are generated for a function which is the sum of a convex part and a concave part, the addition of the two bounding functions is not the tightest possible linear bound for the function. This difficulty

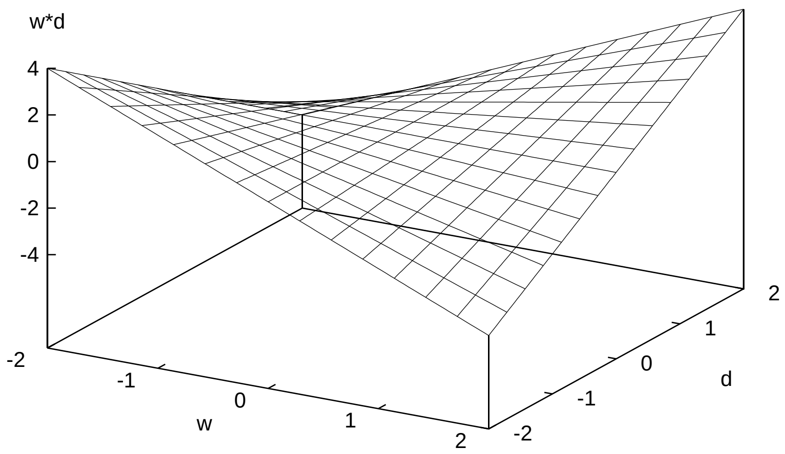
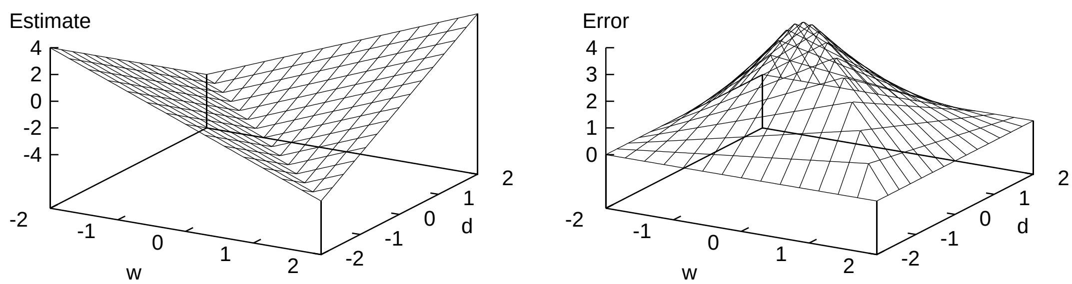


Figure 2: Graph of bilinearity $w\,d$



a) b)

Figure 3: Convex envelope of $w\,d$ and underestimation error

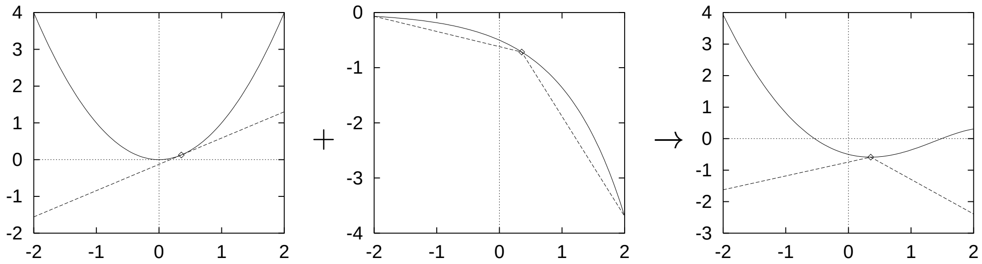


Figure 4: Problem with McCormick's envelopes

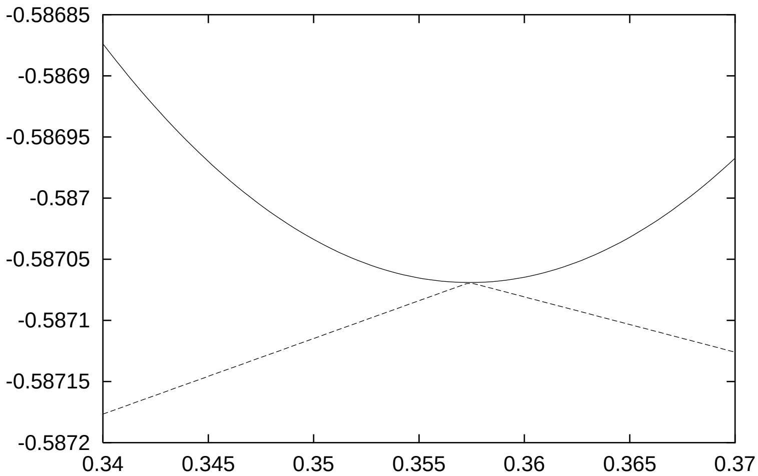


Figure 5: McCormick's envelopes on a small region

is demonstrated in Figure 4, and Figure 5 shows that the problem is not solved by using smaller regions. This difficulty is addressed below in section 3.4.

Both of these difficulties with convex envelopes can be solved after separating the quadratic terms into constraint space contributions and null space contributions. To perform the necessary separation, new coordinate variables $\tilde{d}$ and p are defined for the constraint space and null space respectively. To express the relation between $\tilde{d}$ and p and x, the constraint space basis, G, and the null space basis, N, are defined as follows:

$$G = \left[\cdots, \nabla g_i, \cdots; \cdots, e^{i'}, \cdots\right], \; i \in I_A, i' \in I'_A. \tag{22}$$

Note that $\nabla g_i = \bar{b}^i$ and e^i refers to the i'th column of an $n \times n$ identity matrix. This basis matrix is generally not orthogonal. The basis matrix for the null space, N, can be calculated from G. The rows of G are permuted, so G can be partitioned $G = \begin{bmatrix} G_I \\ G_{II} \end{bmatrix}$ with G_I nonsingular. N may then be taken as

$$N = \begin{bmatrix} G_I^{-T} G_{II}^T \\ -I \end{bmatrix}. \tag{23}$$

This definition forces the columns of N to be orthogonal to those of G as can be seen.

$$G^{\mathrm{T}} N = \begin{bmatrix} G_I^{\mathrm{T}} \; G_{II}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} G_I^{-T} G_{II}^T \\ -I \end{bmatrix} = G_I^{\mathrm{T}} G_I^{-\mathrm{T}} G_{II}^{\mathrm{T}} + -G_{II}^{\mathrm{T}} = 0$$

The relation between the original variables and the transformed coordinate variables is

$$x - \check{x} = G\tilde{d} + Np. \tag{24}$$

By substituting (24) into the quadratic terms of (19), for each $i \in 1, \ldots, m$

$$\begin{aligned}
(x - \check{x})^{\mathrm{T}} \bar{H}^{(i)} (x - \check{x}) &= (G\tilde{d} + Np)^{\mathrm{T}} \bar{H}^{(i)} (G\tilde{d} + Np) \\
&= \tilde{d}^{\mathrm{T}} G^{\mathrm{T}} \bar{H}^{(i)} G\tilde{d} + \tilde{d}^{\mathrm{T}} G^{\mathrm{T}} \bar{H}^{(i)} Np + p^{\mathrm{T}} N^{\mathrm{T}} \bar{H}^{(i)} G\tilde{d} + \\
&\quad p^{\mathrm{T}} N^{\mathrm{T}} \bar{H}^{(i)} Np \\
&= \tilde{d}^{\mathrm{T}} G^{\mathrm{T}} \bar{H}^{(i)} G\tilde{d} + 2p^{\mathrm{T}} N^{\mathrm{T}} \bar{H}^{(i)} G\tilde{d} + p^{\mathrm{T}} N^{\mathrm{T}} \bar{H}^{(i)} Np \\
&= (\overbrace{2G\tilde{d} + 2Np}^{2(x-\check{x})} - G\tilde{d})^{\mathrm{T}} \bar{H}^{(i)} G\tilde{d} + p^{\mathrm{T}} N^{\mathrm{T}} \bar{H}^{(i)} Np.
\end{aligned} \tag{25}$$

By defining,

$$d = G\tilde{d} \;\Leftrightarrow\; \begin{cases} Np + d = x - \check{x} \\ N^{\mathrm{T}} d = 0 \end{cases} \tag{26}$$

and

$$w = 2(x - \check{x}) - d, \tag{27}$$

(25) becomes

$$(x - \check{x})^{\mathrm{T}} \bar{H}^{(i)} (x - \check{x}) = w^{\mathrm{T}} \bar{H}^{(i)} d + p^{\mathrm{T}} N^{\mathrm{T}} \bar{H}^{(i)} N p.$$

Then by defining

$$t_{jk} = p^{\mathrm{T}} (N^{\mathrm{T}} e^{j} e^{k\mathrm{T}} N) p \qquad \forall j, k \in 1, \ldots, n \tag{28}$$

$$q_{jk} = w_j d_k \qquad \forall (j,k) \in 1, \ldots, n \times 1, \ldots, n \tag{29}$$

$$(x - \check{x})^{\mathrm{T}} \bar{H}^{(i)} (x - \check{x}) = \sum_{j,k \in 1,\ldots,n} \bar{H}^{(i)}_{jk} (t_{jk} + q_{jk}) \tag{30}$$

The q_{jk} terms hold the constraint space terms and the cross terms, and the t_{jk} terms hold the null space terms. Linear bounding functions for the q_{jk} and t_{jk} terms will be developed in sections 3.3 and 3.4.

Using definitions (26) and (27), d, w and p can be related directly to $x - \check{x}$ with the following algebra.

$$N^{\mathrm{T}} N p + \overbrace{N^{\mathrm{T}} d}^{0} = N^{\mathrm{T}} (x - \check{x})$$

$$p = (N^{\mathrm{T}} N)^{-1} N^{\mathrm{T}} (x - \check{x})$$

$$N (N^{\mathrm{T}} N)^{-1} N^{\mathrm{T}} (x - \check{x}) + d = x - \check{x}$$

$$d = \left(I - N (N^{\mathrm{T}} N)^{-1} N^{\mathrm{T}} \right) (x - \check{x}) \tag{31}$$

$$\begin{aligned} w &= 2(x - \check{x}) - d \\ &= \left(I + N (N^{\mathrm{T}} N)^{-1} N^{\mathrm{T}} \right) (x - \check{x}) \end{aligned} \tag{32}$$

3.3 Constraint Space Quadratic Terms

Figure 3 demonstrated the problem with using the convex envelope to bound bilinearities when the point of interest is in the interior of the region. This problem will be solved in this section by dividing the region into orthants and using the convex and concave envelope inside of each orthant. The envelopes for all of the orthants will be combined into a single program in chapter 4.

The quadratic terms q_{jk} to be bounded come from the constraint space and cross terms as defined in (29). If $w_j \in [w_j^L, w_j^U]$ and $d_k \in [d_k^L, d_k^U]$, McCormick [66] provides the convex and concave envelopes

$$w_j d_k \geq \max\{d_k^L w_j + w_j^L d_k - d_k^L w_j^L \, , \, d_k^U w_j + w_j^U d_k - d_k^U w_j^U\} \tag{33}$$

$$w_j d_k \leq \min\{d_k^L w_j + w_j^U d_k - d_k^L w_j^U \, , \, d_k^U w_j + w_j^L d_k - d_k^U w_j^L\} \tag{34}$$

The bounds on w and d are calculated by evaluating (32) and (31) with interval arithmetic.

$$\begin{aligned}
[w^L, w^U] &= (I + N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}})([x^L, x^U] - \check{x}) \\
[d^L, d^U] &= (I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}})([x^L, x^U] - \check{x})
\end{aligned}$$

These bounds on w and d superscribe the region $[x^L - \check{x}, x^U - \check{x}]$.

The next step in the bounding procedure is to split the variable region into orthants around $\check{x}$ and then to develop convex and concave envelopes for each orthant. The combined result of this is a piecewise convex/concave envelope. The resulting piecewise convex envelope is demonstrated in Figure 6 along with the estimation error. As desired, there is no estimation error at $\check{x}$.

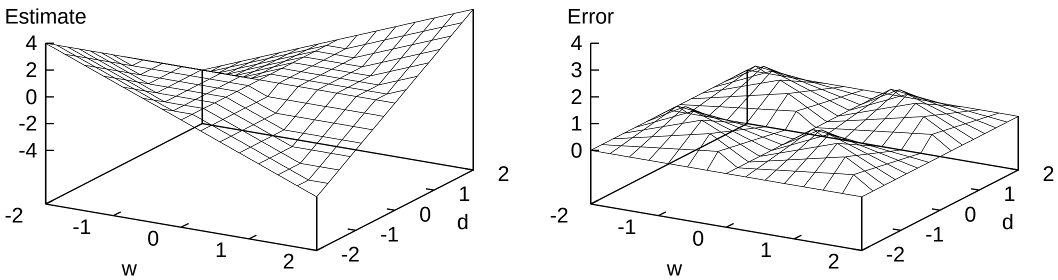


Figure 6: Piecewise convex envelope for bilinearity $w\,d$ and its error

Below are the linear bound constraints that are added to the orthant underestimating program. σ_{jk} indicates the direction of support that is needed.

$$\sigma_{jk} = \operatorname{sign}(\sum_i \bar{u}_i \bar{H}^{(i)}_{jk}) = \operatorname{sign}(\bar{\gamma}_{jk}) \tag{35}$$

$$c^1_{jk} w_j + c^2_{jk} d_k \leq \sigma_{jk} q_{jk} \tag{36}$$

$$c^3_{jk} w_j + c^4_{jk} d_k \leq \sigma_{jk} q_{jk} \tag{37}$$

Let $\left[w'^L, w'^U\right] \times \left[d'^L, d'^U\right]$ be an orthant of $\left[w^L, w^U\right] \times \left[d^L, d^U\right]$. If $\sigma_{jk} = +1$

$$\begin{array}{ll} c^1_{jk} = \begin{cases} d'^L_k, & \text{if } w'^U_j = w^U_j \\ d'^U_k, & \text{if } w'^L_j = w^L_j \end{cases} & c^2_{jk} = 0 \\ c^3_{jk} = 0 & c^4_{jk} = \begin{cases} w'^L_j, & \text{if } d'^U_k = d^U_k \\ w'^U_j, & \text{if } d'^L_k = d^L_k \end{cases} \end{array} \tag{38}$$

If $\sigma_{jk} = -1$

$$\begin{array}{ll} c^1_{jk} = \begin{cases} -d'^U_k, & \text{if } w'^U_j = w^U_j \\ -d'^U_k, & \text{if } w'^L_j = w^L_j \end{cases} & c^2_{jk} = 0 \\ c^3_{jk} = 0 & c^4_{jk} = \begin{cases} -w'^U_j & (d'^U_k = d^U_k) \\ -w'^L_j & (d'^L_k = d^L_k) \end{cases} \end{array} \tag{39}$$

Constraints (36) and (37) have first order gaps in the directions of the space spanned by the gradients of the active constraints, so Lagrange multiplier adjustments will compensate for those deviations in the stationarity conditions. This point will be demonstrated in section 3.5 where the first order optimality conditions for the orthant program will be presented.

3.4 Null Space Quadratic Terms

The remaining quadratic terms are those in the null space directions. These terms cannot be bounded with McCormick's envelopes because, as demonstrated in Figure 4, problems with any concave terms will not be tightly bound even if the combined function behavior is convex. The bounds for these variables are developed from positive definite combinations of the quadratic terms derived from the overall behavior of the program at $\check{x}$ to make a tight bound. The quadratic terms under consideration are defined as follows:

$$t_{jk} = p^{\mathrm{T}}(N^{\mathrm{T}} e^j e^{k\mathrm{T}} N)p \qquad \forall (j,k) \in 1,\ldots,n \times 1,\ldots,n \tag{28}$$

Given any $n \times n$ positive semidefinite (PSD) matrix, γ, the following is true by definition:

$$\begin{aligned} p^{\mathrm{T}} N^{\mathrm{T}} \gamma N p &= p^{\mathrm{T}} \left(\sum_{j,k} \gamma_{jk} (N^{\mathrm{T}} e^j e^{k\mathrm{T}} N) \right) p && (40) \\ &= \sum_{j,k} \gamma_{jk} t_{jk} && (41) \\ &\geq 0 && (42) \end{aligned}$$

Legitimate constraints of this form can be written for any PSD matrix γ, but only certain γ's will give a sufficiently tight bound to support stationarity as desired.

Writing the orthant program with a single constraint of the form (41) provides some insight into the choice of γ. Below is a program with only the constraints involving γ and its first order optimality conditions for the t_{jk} variables using

$$\begin{aligned} \Delta x &= x - \check{x}, \\ \Delta x^U &= x^U - \check{x}, \\ \Delta x^L &= x^L - \check{x}. \end{aligned}$$

$$\begin{aligned} & \min \Delta x_0 \\ & \text{s.t.} \\ u_i \quad : \quad & \bar{g}_i + \bar{b}^{i\mathrm{T}} \Delta x + \sum_{j,k} \hat{H}^{(i)}_{jk}(t_{jk} + q_{jk}) \le 0 \quad \forall\, i \\ \chi \quad : \quad & \sum_{j,k} \bar{\gamma}_{jk} t_{jk} \ge 0 \qquad (43) \\ & \text{First Order Optimality Conditions} \\ t_{jk} \quad : \quad & \sum_i u_i \hat{H}^{(i)}_{jk} - \chi \bar{\gamma}_{jk} = 0 \quad \forall j, k \qquad (44) \end{aligned}$$

If $\check{x}$ is a local minimum of (11), the second order optimality conditions require that $\sum_i \bar{u}_i N^{\mathrm{T}} \bar{H}^{(i)} N$ be PSD, and choosing

$$\bar{\gamma} = \sum_i \bar{u}_i \bar{H}^{(i)} \qquad (45)$$

will satisfy (44) at $\Delta x = 0$ if $\hat{H}^{(i)} = \bar{H}^{(i)}$ and $u = \bar{u}$. However, (43) is not sufficient under general conditions where $\hat{H}^{(i)} \ne \bar{H}^{(i)}$ or $u \ne \bar{u}$. Changes in u may occur because of the gradient differences between the original definition of q_{jk} and its linear bounding functions or due to the interval relaxation explained in chapter 4.

The solution to this difficulty is to generate a set of γ matrices and a corresponding set of constraints to bound the null space quadratic terms. The set of γ's that can accommodate the greatest allowable change in u would inscribe the space of positive definite combinations of the $\hat{H}^{(i)}$'s. It is not clear how that particular set of constraints can be conveniently generated, so the method presented here tries to span the largest space possible while requiring reasonable computational effort and maintaining problem sparsity. The method takes $\bar{\gamma}$ and perturbs it in each of the bilinear directions which appear in the problem until the semi-definite limit is reached. If $\bar{\gamma}$ is not positive definite in the null space, it is adjusted with a diagonal positive definite matrix. Here are the details of the method.

$$Q = N^{\mathrm{T}}\bar{\gamma}N \tag{46}$$

Q is perturbed by one symmetric dyad pair at a time in each direction which appears in the problem.

$$\tilde{Q} = Q - \rho_{jk}N^{\mathrm{T}}(e^j e^{k\mathrm{T}} + e^k e^{j\mathrm{T}})N$$

By realizing that in the semidefinite limit $\tilde{Q}$ becomes singular, it is possible to calculate a formula for the limiting value of ρ_{jk}.

$$\left(Q - \rho_{jk}N^{\mathrm{T}}\left(e^j e^{k\mathrm{T}} + e^k e^{j\mathrm{T}}\right)N\right)x = 0 \;\text{ for some } x \neq 0 \tag{47}$$

$$x = \rho_{jk}\left(Q^{-1}N^{\mathrm{T}}e^j e^{k\mathrm{T}}Nx + Q^{-1}N^{\mathrm{T}}e^k e^{j\mathrm{T}}Nx\right)$$

$$e^{k\mathrm{T}}Nx = \rho_{jk}\left(e^{k\mathrm{T}}NQ^{-1}N^{\mathrm{T}}e^j e^{k\mathrm{T}}Nx + e^{k\mathrm{T}}NQ^{-1}N^{\mathrm{T}}e^k e^{j\mathrm{T}}Nx\right) \tag{48}$$

$$e^{j\mathrm{T}}Nx = \rho_{jk}\left(e^{j\mathrm{T}}NQ^{-1}N^{\mathrm{T}}e^j e^{k\mathrm{T}}Nx + e^{j\mathrm{T}}NQ^{-1}N^{\mathrm{T}}e^k e^{j\mathrm{T}}Nx\right) \tag{49}$$

Then define

$$\begin{aligned} \eta_{jk} &= e^{j\mathrm{T}}NQ^{-1}N^{\mathrm{T}}e^k, \\ a &= e^{k\mathrm{T}}Nx, \\ b &= e^{j\mathrm{T}}Nx. \end{aligned}$$

Substituting these definitions into (48) and (49) gives

$$\begin{aligned} a &= \rho_{jk}\eta_{kj}a + \rho_{jk}\eta_{kk}b \\ b &= \rho_{jk}\eta_{jj}a + \rho_{jk}\eta_{jk}b \end{aligned}$$

Solving these two equations together, gives the limiting values of ρ_{jk}.

$$\rho_{jk}^{-1} = \eta_{jk} \pm \sqrt{\eta_{jj}\eta_{kk}} \tag{50}$$

For each perturbation dyad jk, equation (50) will give either a positive and a zero value for ρ_{jk}^{-1} or two nonzero values of opposite signs. These two values give two constraints each of the form

$$0 \leq -\rho_{jk}(t_{jk} + t_{kj}) + \sum_{r,s} \bar{\gamma}_{rs} t_{rs}. \tag{51}$$

The two values of ρ_{jk}^{-1} are renamed separately as ρ_{jk}^{-1} and $\tilde{\rho}_{jk}^{-1}$, such that

$$\begin{aligned} \rho_{jk}^{-1} &\leq 0, \\ \tilde{\rho}_{jk}^{-1} &\geq 0. \end{aligned}$$

Then the two constraints from (51) can be rewritten as

$$0 \leq (t_{jk} + t_{kj}) + \left|\rho_{jk}^{-1}\right| \sum_{r,s} \bar{\gamma}_{rs} t_{rs} \tag{52}$$

$$0 \leq -(t_{jk} + t_{kj}) + \left|\tilde{\rho}_{jk}^{-1}\right| \sum_{r,s} \bar{\gamma}_{rs} t_{rs}. \tag{53}$$

If $\breve{x}$ is not a local minimum or the Lagrange multipliers are not correct, Q in (46) is potentially indefinite. In the development above, it is necessary to factorize Q. A modified Cholesky algorithm [107] is used, and if necessary, a diagonal matrix is added to make Q positive definite. $\bar{\gamma}$ is modified to incorporate the adjustment to

Q. Given an diagonal adjustment E to Q, the adjustment to γ can be determined as follows:

$$\begin{aligned} N^{\mathrm{T}}(\bar{\gamma} + \begin{bmatrix} 0 & 0 \\ 0 & E \end{bmatrix})N &= N^{\mathrm{T}}\bar{\gamma}N + \\ & \begin{bmatrix} G_{II}G_I^{-1} & -I \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 0 & 0 \\ 0 & E \end{bmatrix} \begin{bmatrix} G_I^{-T}G_{II}^T \\ -I \end{bmatrix} \\ &= Q + E \end{aligned} \tag{54}$$

Constraints (52) and (53) are included in the orthant program for every pair jk that is used in the problem or added by the diagonal adjustment. When $\left|\left|\hat{H}^{(i)} - \bar{H}^{(i)}\right|\right|$ and $||u - \bar{u}||$ are not too large, this set of constraints derived from perturbing Q in each direction is sufficient to support the first order optimality conditions of (11).

3.5 Orthant Program

With the pieces derived, the complete orthant program can be written. This orthant program is useful as an intermediate result; though there are too many orthants, 2^n, to be used directly in an algorithm. This combinatorial difficulty is solved by the covering program presented in the next chapter.

It is illuminating to compare the original quadratic NLP with the orthant program and to compare their first order optimality conditions.

Quadratic NLP	**Orthant Program**

$$
\begin{array}{llll}
 & \min \Delta x_0 & \min \Delta x_0 & (55) \\
 & \text{s.t.} & \text{s.t.} & \\
u_i: & \bar{g}_i + \bar{b}^{i\mathrm{T}}\Delta x + \sum_{j,k} \hat{H}^{(i)}_{jk}(t_{jk}+q_{jk}) \le 0 \quad \forall i & \bar{g}_i + \bar{b}^{i\mathrm{T}}\Delta x + \sum_{j,k} \hat{H}^{(i)}_{jk}(t_{jk}+q_{jk}) \le 0 \quad \forall i & (56) \\
\chi_{jk}: & t_{jk} = p^{\mathrm{T}} N^{\mathrm{T}} e^j e^{k\mathrm{T}} N p \quad \forall j,k & \begin{cases} 0 \le t_{jk} + t_{kj} + |\rho^{-1}_{jk}| \sum_{r,s} \bar{\gamma}_{rs} t_{rs} & \text{(A)} \\ 0 \le -(t_{jk}+t_{kj}) + |\tilde{\rho}^{-1}_{jk}| \sum_{r,s} \bar{\gamma}_{rs} t_{rs} & \text{(B)} \end{cases} \quad \forall j,k & (57) \\
\alpha: & q_{jk} = w_j d_k \quad \forall j,k & \begin{cases} c^1_{jk} w_j \le \sigma_{jk} q_{jk} & \text{(A)} \\ c^4_{jk} d_k \le \sigma_{jk} q_{jk} & \text{(B)} \end{cases} \quad \forall j,k & (58) \\
\beta: & Np + d = \Delta x & Np + d = \Delta x & (59) \\
\varphi: & N^{\mathrm{T}} d = 0 & N^{\mathrm{T}} d = 0 & (60) \\
\pi: & w = 2\Delta x - d & w = 2\Delta x - d & (61) \\
\lambda: & \Delta x^L \le \Delta x \le \Delta x^U & \Delta x^L \le \Delta x \le \Delta x^U & (62) \\
\kappa: & w'^L \le w \le w'^U & w'^L \le w \le w'^U & (63) \\
\mu: & d'^L \le d \le d'^U & d'^L \le d \le d'^U & (64) \\
 & & & (65)
\end{array}
$$

Their first order optimality conditions compare as follows:

	First Order Conditions	**First Order Conditions**	
$\Delta x:$	$e^0 + \sum_i u_i \bar{b}^i - \beta - 2\pi + \lambda = 0$	$e^0 + \sum_i u_i \bar{b}^i - \beta - 2\pi + \lambda = 0$	(66)

$$
\begin{array}{rll}
t_{jk}: & \sum_i u_i \hat{H}^{(i)}_{jk} + \chi_{jk} = 0 & \sum_i u_i \hat{H}^{(i)}_{jk} - \chi^A_{jk} - \chi^A_{kj} + \\
 & & \chi^B_{jk} + \chi^B_{kj} - \\
 & & \bar{\gamma}_{jk} \sum_{r,s} (\chi^A_{rs} |\rho^{-1}_{rs}| + \chi^B_{rs} |\tilde{\rho}^{-1}_{rs}|) = 0 \qquad (67) \\
q_{jk}: & \sum_i u_i \hat{H}^{(i)}_{jk} + \alpha_{jk} = 0 & \sum_i u_i \hat{H}^{(i)}_{jk} - \sigma_{jk}(\alpha^{\mathrm{A}}_{jk} + \alpha^{\mathrm{B}}_{jk}) = 0 \qquad (68) \\
p: & -\sum_{j,k} 2\chi_{jk} N^{\mathrm{T}} e^j e^{k\mathrm{T}} N p + & N^{\mathrm{T}} \beta = 0 \qquad (69) \\
 & N^{\mathrm{T}} \beta = 0 & \\
w_j: & -\sum_k \alpha_{jk} d_k + \pi_j + \kappa_j = 0 & \sum_k \alpha^{\mathrm{A}}_{jk} c^1_{jk} + \pi_j + \kappa_j = 0 \qquad (70) \\
d_k: & -\sum_j \alpha_{jk} w_j + \beta_k + \pi_k + & \sum_k \alpha^{\mathrm{B}}_{jk} c^4_{jk} + \beta_k + \qquad (71) \\
 & N^{\mathrm{T}}_k \varphi + \mu_k = 0 & \pi_k + N^{\mathrm{T}}_k \varphi + \mu_k = 0 \\
 & u \geq 0 & u, \alpha^{\mathrm{A}}, \alpha^{\mathrm{B}}, \chi^A, \chi^B \geq 0 \qquad (72)
\end{array}
$$

For the most part, the programs are very similar, differing only in how the quadratic variables are treated, and if $\Delta x = 0$ satisfies the original program's optimality conditions, $\Delta x = 0$ will also satisfy the optimality conditions of the orthant program if the region is small enough. The changes in (58) create differences in (68), (70), and (71). Constraint (68) can be satisfied by adjusted values of α^A_{jk} and α^B_{jk} as long as $\mathrm{sign}(\bar{\gamma}_{jk}) = \mathrm{sign}(\sum_i u_i \hat{H}^{(i)}_{jk})$. These changes in α will ultimately require changes in u via the variables π and β.

The changes in (57) create differences in (67) and (69). When $\Delta x = 0$, constraints (69) are identical because $p = 0$, so that will not cause the orthant program to have a different solution. The orthant version of (67) may be satisfied by some combination of χ^A_{jk}, χ^B_{jk} if $\sum_i u_i \hat{H}^{(i)}$ is in the space of positive semi-definite matrices spanned by the set of γ developed above which will be true when $||\bar{\gamma} - \sum_i u_i \hat{H}^{(i)}||$ is not too large.

Chapter 4

The Covering Program

The orthant program derived in the previous chapter is impractical as a means of obtaining a bound on a region because it implies the solving of 2^n subproblems. In this chapter, an interval-based relaxation will be developed and then applied to the set of 2^n orthant programs to give a single program to solve for a lower bound on a region.

In the orthant program, only constraints (58.A) and (58.B) and variable bounds (63) and (64) are affected by the choice of orthant. To extend the orthant program into a single program for the whole variable range, the variable bounds can be changed to the whole range, and the coefficients in constraints (58.A) and (58.B) can be replaced with intervals.

4.1 Constraints from an Interval Linear System

For the derivation that follows, it is necessary to develop a set of linear constraints from a linear system of equations with interval coefficients. Given a linear system

$$Au = c \tag{73}$$
$$u \geq 0$$

with $A \in [\underline{A}, \bar{A}]$, an interval matrix, the goal is to develop the tightest set of linear constraints that can be written to limit the range of u, $[\underline{u}, \bar{u}]$. Use of LP and linear constraints to approximate linear systems with interval coefficients has been studied before [15, 17, 77, 78]. The method presented here includes the constraints of this previous work plus an additional constraint to increase the tightness of the bounds on u. Neumaier [76] presents other interval methods for solving this problem, but these methods are not applicable for the method presented here because they cannot be used within a LP.

For any variable x, $x^{(+)}$ refers to its positive part, and $x^{(-)}$ refers to its negative part. It is formally defined as follows:

$$\begin{aligned} x^{(+)} &= \max\{0, x\} \\ x^{(-)} &= \max\{0, -x\} \end{aligned}$$

Consider a row i of (73), and choose a particular multiplier, $u_{r(i)}$. Under the condition that $u \geq 0$, the following condition may be used to obtain one limit of a valid interval $[\underline{u}_{r(i)}, \bar{u}_{r(i)}]$ containing the value of $u_{r(i)}$ for some solution of (73) for all $A_{i\,r(i)} \in [\underline{A}_{i\,r(i)}, \bar{A}_{i\,r(i)}]$

$$A_{i\,r(i)} u_{r(i)} \leq c_i - \sum_{j \neq r(i)} (\bar{A}_{ij}^{(+)} \bar{u}_j - \bar{A}_{ij}^{(-)} \underline{u}_j) \tag{74}$$

One method for selecting $r(i)$ will be described in the following section. By choosing the value of $A_{i\,r(i)}$ which maximizes the left hand side and the value of $u_{r(i)}$ which minimizes it, the following is obtained.

$$\bar{A}^{(+)}_{i\,r(i)}\underline{u}_{r(i)} - \bar{A}^{(-)}_{i\,r(i)}\bar{u}_{r(i)} \leq c_i - \sum_{j \neq r(i)} (\bar{A}^{(+)}_{ij}\bar{u}_j - \bar{A}^{(-)}_{ij}\underline{u}_j) \tag{75}$$

This can be rewritten as

$$\sum_j (\bar{A}^{(+)}_{ij}\bar{u}_j - \bar{A}^{(-)}_{ij}\underline{u}_j) - |\bar{A}_{i\,r(i)}|(\bar{u}_{r(i)} - \underline{u}_{r(i)}) \leq c_i \tag{76}$$

Similarly, the following condition may also be required

$$A_{i\,r(i)}u_{r(i)} \geq c_i - \sum_{j \neq r(i)} (\underline{A}^{(+)}_{ij}\underline{u}_j - \underline{A}^{(-)}_{ij}\bar{u}_j) \tag{77}$$

for all $A_{i\,r(i)} \in [\underline{A}_{i\,r(i)}, \bar{A}_{i\,r(i)}]$ and some $u_{r(i)}$. By choosing the $A_{i\,r(i)}$ which minimizes the left hand size and the value of $u_{r(i)}$ which maximizes it and performing similar rearrangements, the following is similarly obtained.

$$\sum_j (\underline{A}^{(-)}_{ij}\bar{u}_j - \underline{A}^{(+)}_{ij}\underline{u}_j) - |\underline{A}_{i\,r(i)}|(\bar{u}_{r(i)} - \underline{u}_{r(i)}) \leq -c_i \tag{78}$$

Combining (76) and (78) for all i gives a set of constraints which can be used to determine intervals $[\underline{u}, \bar{u}]$ containing the values of the solution u to (73) for $A \in [\underline{A}, \bar{A}]$. These constraints will be used below to develop the underestimating linear program.

4.2 Choosing $r(i)$

In the previous section, $r(i)$ was an unspecified function assigning a particular variable for each row. In this section, the method of assignment developed by

Swaney [114] will be presented. The choice of $r(i)$ can greatly affect the quality of the constraints developed in the previous section.

The following method requires an approximate solution, $\check{u}$ of the interval linear system, (73), for some value of $A \in [\underline{A}, \bar{A}]$. $\check{u}$ can be taken from a previous iteration or can be initialized to a vector of ones.

The function $r(i)$ is defined in sequential fashion. First a row i is chosen from the list of unassigned rows, and then the column for that row, $r(i)$ is chosen from the list of unassigned columns. This continues until all rows have been assigned. The following should be calculated for use in choosing the rows and columns.

$$\bar{\chi} = \bar{A}\check{u} \tag{79}$$

$$\underline{\chi} = -\underline{A}\check{u} \tag{80}$$

$$i^{\text{next}} = \underset{\text{unassigned } i}{\arg\max} \left(\min_{\text{unassigned } j} \max \left\{ \left(\frac{\bar{\chi}_i}{\bar{A}_{ij}\check{u}_j} \right), \left(\frac{\underline{\chi}_i}{-\underline{A}_{ij}\check{u}_j} \right) \right\} \right) \tag{81}$$

$$r(i) = \underset{\text{unassigned } j}{\arg\min} \max \left\{ \left(\frac{\bar{\chi}_i}{\bar{A}_{ij}\check{u}_j} \right), \left(\frac{\underline{\chi}_i}{-\underline{A}_{ij}\check{u}_j} \right) \right\} \qquad i = i^{\text{next}} \tag{82}$$

$r(i)$ is defined by calculating i^{next} and then calculating $r(i^{\text{next}})$. Let $s(j)$ be defined as the inverse function of $r(i)$ such that

$$\begin{aligned} i &= s(r(i)) \\ j &= r(s(j)) \end{aligned}$$

4.3 Interval Linear Program

Now that constraints for a interval linear system have been developed, it is possible to develop an underestimating program for a linear program that has an interval coefficient matrix. Given an LP of the form:

$$P(A) = \min c^{\mathrm{T}} x \tag{83}$$
$$\text{s.t. } Ax \leq b$$

with $A \in [\underline{A}, \bar{A}] \subset IR^{m \times n}$, the set of all $m \times n$ matrices with interval coefficients. The dual of this LP is:

$$D(A) = \min b^{\mathrm{T}} u \tag{84}$$
$$A^{\mathrm{T}} u = -c \tag{85}$$
$$u \geq 0 \tag{86}$$

Constraint (85) is an interval linear system, so the bounding constraints developed in Section 4.1 above can be applied to it. Applying these constraints leads to the following linear program

$$\min \quad [b^{(+)\mathrm{T}}, -b^{(-)\mathrm{T}}] \begin{bmatrix} \bar{u} \\ \underline{u} \end{bmatrix} \tag{87}$$
$$\text{s.t.} \quad \sum_j \left(\underline{A}_{ji}^{(-)} \bar{u}_j - \underline{A}_{ji}^{(+)} \underline{u}_j \right) - |\underline{A}_{r(i)i}| \left(\bar{u}_{r(i)} - \underline{u}_{r(i)} \right) \leq c_i \qquad \forall i \tag{88}$$
$$\sum_j \left(\bar{A}_{ji}^{(+)} \bar{u}_j - \bar{A}_{ji}^{(-)} \underline{u}_j \right) - |\bar{A}_{r(i)i}| \left(\bar{u}_{r(i)} - \underline{u}_{r(i)} \right) \leq -c_i \quad \forall i \tag{89}$$
$$-\underline{u} \leq 0 \tag{90}$$
$$\underline{u} - \bar{u} \leq 0 \tag{91}$$

Constraints (88) and (89) define restrictions on $\underline{u}, \bar{u}$ such that they describe a valid interval containing the solution to (84) for all $A \in [\underline{A}, \bar{A}]$. The interval $[\underline{u}, \bar{u}]$ so defined may overestimate the true range of u values in (84), i.e. these constraints may be somewhat overrestrictive in that role. Also, (90) is introduced as an added restriction. Thus this LP representation of the interval dual will be referred to as the restricted dual. The choice for the objective function will be justified below. The order of the subscripts of A reflects the fact that constraint (85) uses A^{T}.

Next the restricted dual is transformed back to corresponding primal variables.

$$\min \quad c^{\mathrm{T}}\left(x^{(+)} - x^{(-)}\right) \tag{92}$$

$$\text{s.t.} \quad \begin{aligned} &\sum_j \left(\underline{A}_{ij}^{(-)} x_j^{(+)} + \bar{A}_{ij}^{(+)} x_j^{(-)}\right) - \\ &|\underline{A}_{i\,s(i)}| x_{s(i)}^{(+)} - |\bar{A}_{i\,s(i)}| x_{s(i)}^{(-)} - y_i = -b_i^{(+)} \qquad \forall i \end{aligned} \tag{93}$$

$$\begin{aligned} &\sum_j \left(-\underline{A}_{ij}^{(+)} x_j^{(+)} - \bar{A}_{ij}^{(-)} x_j^{(-)}\right) + \\ &|\underline{A}_{i\,s(i)}| x_{s(i)}^{(+)} + |\bar{A}_{i\,s(i)}| x_{s(i)}^{(-)} - z_i + y_i = b_i^{(-)} \qquad \forall i \end{aligned} \tag{94}$$

$$x^{(+)} \geq 0 \tag{95}$$

$$x^{(-)} \geq 0 \tag{96}$$

$$z \geq 0 \tag{97}$$

$$y \geq 0 \tag{98}$$

Equation (93) can be used to eliminate y, and z converts (94) into an inequality.

$$\begin{aligned} \min \quad & c^{\mathrm{T}}\left(x^{(+)} - x^{(-)}\right) \\ \text{s.t.} \quad & \sum_j \left(\underline{A}_{ij}^{(+)} - \underline{A}_{ij}^{(-)}\right) x_j^{(+)} - \left(\bar{A}_{ij}^{(+)} - \bar{A}_{ij}^{(-)}\right) x_j^{(-)} \leq b_i \qquad \forall i \\ & |\underline{A}_{i\,s(i)}| x_{s(i)}^{(+)} + |\bar{A}_{i\,s(i)}| x_{s(i)}^{(-)} - \sum_j \left(\underline{A}_{ij}^{(-)} x_j^{(+)} + \bar{A}_{ij}^{(+)} x_j^{(-)}\right) \leq b_i^{(+)} \quad \forall i \\ & x^{(+)} \geq 0 \\ & x^{(-)} \geq 0 \end{aligned}$$

Simplifying gives the underestimating program:

$$\min \quad c^{\mathrm{T}}\left(x^{(+)} - x^{(-)}\right) \tag{99}$$

$$\text{s.t.} \quad \underline{A}x^{(+)} - \bar{A}x^{(-)} \leq b \tag{100}$$

$$|\underline{A}_{i\,s(i)}|x^{(+)}_{s(i)} + |\bar{A}_{i\,s(i)}|x^{(-)}_{s(i)} - \sum_j \left(\underline{A}^{(-)}_{ij}x^{(+)}_j + \bar{A}^{(+)}_{ij}x^{(-)}_j\right) \leq b^{(+)}_i \quad \forall i \tag{101}$$

$$x^{(+)} \geq 0 \tag{102}$$

$$x^{(-)} \geq 0 \tag{103}$$

The choice of the objective function (87) for the restricted dual causes the right hand side of (100) to be equivalent to the constraints in the original LP, and it makes (101) as tight as it can be.

Summarizing, for constraints with interval coefficients of the form

$$d \leq [\underline{B}, \bar{B}]x, \tag{104}$$

the corresponding set of LP constraints are

$$d \leq \bar{B}x^{(+)} - \underline{B}x^{(-)} \tag{105}$$

$$|\bar{B}_{i\,s(i)}|x^{(+)}_{s(i)} + |\underline{B}_{i\,s(i)}|x^{(-)}_{s(i)} - \sum_j \left(\bar{B}^{(+)}_{ij}x^{(+)}_j + \underline{B}^{(-)}_{ij}x^{(-)}_j\right) \leq d^{(-)}_i \tag{106}$$

Equality constraints in the original program can be treated equivalently as a pair of opposing inequality constraints using the above.

4.4 Proving the Underestimating Program

The objective function of (99) is identical to (83). To establish that (99) provides a lower bound to the original linear program, (83), it must be shown that every feasible point of (83) has a feasible counterpart in (99). Given $\tilde{x}$, a feasible point of (83), it will be shown that

$$x^{(+)} = \tilde{x}^{(+)}$$
$$x^{(-)} = \tilde{x}^{(-)}$$

is feasible in (99). This choice of $x^{(+)}$ and $x^{(-)}$ clearly satisfies the positivity constraints (102) and (103). The following shows that (100) is satisfied:

$$\begin{aligned} \underline{A}x^{(+)} - \bar{A}x^{(-)} &= (\underline{A} - A)x^{(+)} + (A - \bar{A})x^{(-)} + A(x^{(+)} - x^{(-)}) \\ &= \underbrace{(\underline{A} - A)x^{(+)}}_{\leq 0} + \underbrace{(A - \bar{A})x^{(-)}}_{\leq 0} + \underbrace{A\tilde{x}}_{\leq b} \\ &\leq b \end{aligned}$$

To justify the right hand side of (101) and to illustrate that it is satisfied, it is useful to consider a simple constraint

$$x_j \leq b_i$$

Assuming that $s(i) = j$, applying constraint (101) to this gives

$$x_j^{(+)} \leq b_i^{(+)}$$

If b_i were used in place of $b_i^{(+)}$, the underestimating program would be infeasible when $b_i < 0$ because $x_j^{(+)} \geq 0$ and $x_j \leq b_i < 0$. When $b_i \geq 0$, it is the tightest upper bound for (101). Anything less than b_i, would be more restrictive than the original constraint, and anything greater would be needlessly loose.

To prove that (101) is satisfied by $\tilde{x}$, two cases need to be considered. In the first case, $s(i) = 0$ or $A_{i\,s(i)} = 0$, so (101) is reduced to

$$\begin{aligned} -\sum_j \underbrace{(\underline{A}_{ij}^{(-)} x_j^{(+)} + \bar{A}_{ij}^{(+)} x_j^{(-)})}_{\geq 0} &\leq 0 \\ &\leq b_i^{(+)} \end{aligned}$$

In this case, this constraint is redundant, so it need not be included when solving the underestimating LP.

In the remaining case, $s(i) \neq 0$, and $A_{i\,s(i)} \neq 0$. From above, the following is true for row i.

$$\begin{aligned}
b_i^{(+)} &\geq b_i \\
&\geq \underline{A}_i^{\mathrm{T}} x^{(+)} - \bar{A}_i^{\mathrm{T}} x^{(-)} \\
&= \sum_j ((\underline{A}_{ij}^{(+)} - \underline{A}_{ij}^{(-)}) x_j^{(+)} - (\bar{A}_{ij}^{(+)} - \bar{A}_{ij}^{(-)}) x_j^{(-)}) \\
&= (\underline{A}_{i\,s(i)}^{(+)} - \underline{A}_{i\,s(i)}^{(-)}) x_{s(i)}^{(+)} - (\bar{A}_{i\,s(i)}^{(+)} - \bar{A}_{i\,s(i)}^{(-)}) x_{s(i)}^{(-)} + \\
&\quad \sum_{j \neq s(i)} ((\underbrace{\underline{A}_{ij}^{(+)} x_j^{(+)} + \bar{A}_{ij}^{(-)} x_j^{(-)}}_{\geq 0}) - (\underline{A}_{ij}^{(-)} x_j^{(+)} + \bar{A}_{ij}^{(+)} x_j^{(-)})) \\
&\geq \underline{A}_{i\,s(i)}^{(+)} x_{s(i)}^{(+)} + \bar{A}_{i\,s(i)}^{(-)} x_{s(i)}^{(-)} - \underline{A}_{i\,s(i)}^{(-)} x_{s(i)}^{(+)} - \bar{A}_{i\,s(i)}^{(+)} x_{s(i)}^{(-)} - \\
&\quad \sum_{j \neq s(i)} (\underline{A}_{ij}^{(-)} x_j^{(+)} + \bar{A}_{ij}^{(+)} x_j^{(-)})
\end{aligned}$$

Because of the $(+)$ and $(-)$ designations and the fact that $\underline{A} \leq \bar{A}$, only one of the four products involving $A_{i\,s(i)}$ and $x_{s(i)}$ may be nonzero. This constraint must be satisfied for each of these four terms alone. The third and fourth terms are satisfied trivially, and the first and second terms give the following constraint which is equivalent to (101).

$$b_i^{(+)} \geq \underline{A}_{i\,s(i)}^{(+)} x_{s(i)}^{(+)} + \bar{A}_{i\,s(i)}^{(-)} x_{s(i)}^{(-)} - \sum_{j \neq s(i)} (\underline{A}_{ij}^{(-)} x_j^{(+)} + \bar{A}_{ij}^{(+)} x_j^{(-)})$$

Thus, every feasible point in the original problem has a corresponding solution to the underestimating program.

4.5 Modifying the Orthant Program

To finish the derivation of the covering program, an interval form of the orthant program must be developed and the interval relaxation applied. In addition to the interval changes, several modifications are made to the orthant program to improve its bounding behavior.

The constraints (58.A) and (58.B) are the only general constraints to depend on the choice of orthant. The values of c^1_{jk} and c^4_{jk} are determined by equations (38) and (39). If $\sigma_{jk} = 1$, $c^1_{jk} \in [d^L_k, d^U_k]$, and $c^4_{jk} \in [w^L_j, w^U_j]$. The convex and concave envelopes of these quadratic terms can be written as (109) and (110). The simple variable bounds on w and d become (121) and (122).

The general McCormick convex and concave envelopes for $\Delta x_j \Delta x_k$ are added to the program to give a bound which will work if the other bounding functions fail to keep the LP stationary at $\Delta x = 0$. This requires the addition of constraints (111–114).

$$\min \quad c^{\mathrm{T}} \Delta x \tag{107}$$

$$\text{s.t.} \quad \bar{g}_i + \bar{b}^{i\mathrm{T}} \Delta x + \sum_{j,k} \hat{H}^{(i)}_{jk}(t_{jk} + q_{jk}) \leq 0 \qquad \forall i \tag{108}$$

$$0 \leq [d^L_k, d^U_k] w_j - q_{jk} \leq 0 \qquad \forall j,k \tag{109}$$

$$0 \leq [w^L_k, w^U_k] d_j - q_{jk} \leq 0 \qquad \forall j,k \tag{110}$$

$$\Delta x^U_j \Delta x_k + \Delta x^U_k \Delta x_j - \tfrac{1}{2}(q_{jk} + t_{jk} + q_{kj} + t_{kj}) \leq \Delta x^U_j \Delta x^U_k \qquad \forall j,k \tag{111}$$

$$\Delta x^L_j \Delta x_k + \Delta x^L_k \Delta x_j - \tfrac{1}{2}(q_{jk} + t_{jk} + q_{kj} + t_{kj}) \leq \Delta x^L_j \Delta x^L_k \qquad \forall j,k \tag{112}$$

$$\Delta x^L_j \Delta x_k + \Delta x^U_k \Delta x_j - \tfrac{1}{2}(q_{jk} + t_{jk} + q_{kj} + t_{kj}) \geq \Delta x^L_j \Delta x^U_k \qquad \forall j,k \tag{113}$$

$$\Delta x^U_j \Delta x_k + \Delta x^L_k \Delta x_j - \tfrac{1}{2}(q_{jk} + t_{jk} + q_{kj} + t_{kj}) \geq \Delta x^U_j \Delta x^L_k \qquad \forall j,k \tag{114}$$

$$0 \leq \quad (t_{jk} + t_{kj}) + |\rho_{jk}^{-1}| \sum_{r,s} \bar{\gamma}_{rs} t_{rs} \qquad \forall j,k \tag{115}$$

$$0 \leq -(t_{jk} + t_{kj}) + |\tilde{\rho}_{jk}^{-1}| \sum_{r,s} \bar{\gamma}_{rs} t_{rs} \qquad \forall j,k \tag{116}$$

$$Np + d = \Delta x \tag{117}$$

$$N^{\mathrm{T}} d = 0 \tag{118}$$

$$w + d = 2x \tag{119}$$

$$\Delta x^L \leq \Delta x \leq \Delta x^U \tag{120}$$

$$w^L \leq w \leq w^U \tag{121}$$

$$d^L \leq d \leq d^U \tag{122}$$

Equations (117)–(119) are used to eliminate w and d from this formulation giving:

$$\min \ c^{\mathrm{T}} \Delta x \tag{123}$$

$$\text{s.t. } \bar{g}_i + \bar{b}^{i\mathrm{T}} \Delta x + \sum_{j,k} \hat{H}_{jk}^{(i)} (t_{jk} + q_{jk}) \leq 0 \qquad \forall i \tag{124}$$

$$0 \leq [d_k^L, d_k^U] e^{j\mathrm{T}} (I + N(N^{\mathrm{T}} N)^{-1} N^{\mathrm{T}}) \Delta x - q_{jk} \leq 0 \qquad \forall j,k \tag{125}$$

$$0 \leq [w_k^L, w_k^U] e^{j\mathrm{T}} (I - N(N^{\mathrm{T}} N)^{-1} N^{\mathrm{T}}) \Delta x - q_{jk} \leq 0 \qquad \forall j,k \tag{126}$$

$$\Delta x_j^U \Delta x_k + \Delta x_k^U \Delta x_j - \tfrac{1}{2}(q_{jk} + t_{jk} + q_{kj} + t_{kj}) \leq \Delta x_j^U \Delta x_k^U \quad \forall j,k \tag{127}$$

$$\Delta x_j^L \Delta x_k + \Delta x_k^L \Delta x_j - \tfrac{1}{2}(q_{jk} + t_{jk} + q_{kj} + t_{kj}) \leq \Delta x_j^L \Delta x_k^L \quad \forall j,k \tag{128}$$

$$\Delta x_j^L \Delta x_k + \Delta x_k^U \Delta x_j - \tfrac{1}{2}(q_{jk} + t_{jk} + q_{kj} + t_{kj}) \geq \Delta x_j^L \Delta x_k^U \quad \forall j,k \tag{129}$$

$$\Delta x_j^U \Delta x_k + \Delta x_k^L \Delta x_j - \tfrac{1}{2}(q_{jk} + t_{jk} + q_{kj} + t_{kj}) \geq \Delta x_j^U \Delta x_k^L \quad \forall j,k \tag{130}$$

$$0 \leq \quad (t_{jk} + t_{kj}) + |\rho_{jk}^{-1}| \sum_{r,s} \bar{\gamma}_{rs} t_{rs} \qquad \forall j,k \tag{131}$$

$$0 \leq -(t_{jk} + t_{kj}) + |\tilde{\rho}_{jk}^{-1}| \sum_{r,s} \bar{\gamma}_{rs} t_{rs} \qquad \forall j,k \tag{132}$$

$$\Delta x^L \leq \Delta x \leq \Delta x^U \tag{133}$$

When the interval LP relaxation derived above is applied to this LP, it is able to verify the global minimum for some finite region when there is no null space. However, it is unable to verify the global minimum for finite sized regions when there is a null space. This problem can be solved by introducing a new variable z to deal with the rank deficiency of G.

z is defined by these two conditions.

$$z + N\tilde{p} = \Delta x \tag{134}$$

$$z_j = 0 \quad j \notin J \tag{135}$$

Inserting the definition of N and partitioning z and Δx according to J gives the following results.

$$\left.\begin{array}{r} \begin{bmatrix} z^{\mathrm{I}} \\ z^{\mathrm{II}} \end{bmatrix} + \begin{bmatrix} G_{\mathrm{I}}^{-\mathrm{T}} G_{\mathrm{II}}^{\mathrm{T}} \\ -I \end{bmatrix} \tilde{p} = \begin{bmatrix} \Delta x^{\mathrm{I}} \\ \Delta x^{\mathrm{II}} \end{bmatrix} \\ \\ z^{\mathrm{II}} = 0 \end{array}\right\} \Rightarrow \tilde{p} = -\Delta x^{\mathrm{II}}$$

$$z = (I + [0\ N])\Delta x \tag{136}$$

Bounds on z are calculated by evaluating (136) using interval arithmetic.

$$[z^L, z^U] = (I + [0\ N])[\Delta x^L, \Delta x^U] \tag{137}$$

z has two important properties which allow it to replace Δx in (124) for $i \in I_A$ and (126). First,

$$\bar{b}^{i\mathrm{T}} z = \bar{b}^{i\mathrm{T}} \Delta x \quad \forall i \in I_A$$

because $\bar{b}^{i\mathrm{T}} N = 0$ for $i \in I_A$. Second, the components of z and Δx in the constraint space are identical.

$$\begin{aligned}
(I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}})\Delta x &= (I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}})z \\
&= (I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}})(I + [0\ N])\Delta x \\
&= (I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}})\Delta x + \\
&\quad \overbrace{(I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}})\,[0\ N]}^{0}\,\Delta x
\end{aligned}$$

Introducing z into the LP and replacing Δx with it where possible tightens the bounds produced by the covering program.

An additional variable change is needed to improve the covering program's bounds. It is possible to reduce the number of t variables by enforcing the symmetry $t_{jk} = t_{kj}$. This is valid because of the definition of t in (28). However, q_{jk} is not generally equal to q_{kj}, so similar treatment for q is not possible. When using the LP presented below, only variables t_{jk} with $j \leq k$ are included, and whenever t_{jk} with $j > k$ appears, t_{kj} is used in its place.

$$\begin{aligned}
\min \quad & c^{\mathrm{T}}\Delta x && && (138)\\
\text{s.t.} \quad & \bar{b}^{i\mathrm{T}} z + \sum_{j,k} \hat{H}^{(i)}_{jk}(t_{jk} + q_{jk}) \leq 0 && \forall i \in I_A && (139)\\
& \bar{g}_i + \bar{b}^{i\mathrm{T}}\Delta x + \sum_{j,k} \hat{H}^{(i)}_{jk}(t_{jk} + q_{jk}) \leq 0 && \forall i \notin I_A && (140)\\
& 0 \leq [d^L_k, d^U_k] e^{j\mathrm{T}}(I + N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}})\Delta x - q_{jk} \leq 0 && \forall j,k && (141)\\
& 0 \leq [w^L_k, w^U_k] e^{j\mathrm{T}}(I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}})z - q_{jk} \leq 0 && \forall j,k && (142)\\
& \Delta x^U_j \Delta x_k + \Delta x^U_k \Delta x_j - \tfrac{1}{2}(q_{jk} + q_{kj} + 2t_{jk}) \leq \Delta x^U_j \Delta x^U_k && \forall j,k && (143)
\end{aligned}$$

$$\Delta x_j^L \Delta x_k + \Delta x_k^L \Delta x_j - \tfrac{1}{2}(q_{jk} + q_{kj} + 2t_{jk}) \leq \Delta x_j^L \Delta x_k^L \quad \forall j,k \tag{144}$$

$$\Delta x_j^L \Delta x_k + \Delta x_k^U \Delta x_j - \tfrac{1}{2}(q_{jk} + q_{kj} + 2t_{jk}) \geq \Delta x_j^L \Delta x_k^U \quad \forall j,k \tag{145}$$

$$\Delta x_j^U \Delta x_k + \Delta x_k^L \Delta x_j - \tfrac{1}{2}(q_{jk} + q_{kj} + 2t_{jk}) \geq \Delta x_j^U \Delta x_k^L \quad \forall j,k \tag{146}$$

$$0 \leq \quad 2t_{jk} + |\rho_{jk}^{-1}| \sum_{r,s} \bar{\gamma}_{rs} t_{rs} \quad \forall j,k \tag{147}$$

$$0 \leq -2t_{jk} + |\tilde{\rho}_{jk}^{-1}| \sum_{r,s} \bar{\gamma}_{rs} t_{rs} \quad \forall j,k \tag{148}$$

$$z = \Delta x + N\Delta x \tag{149}$$

$$\Delta x^L \leq \Delta x \leq \Delta x^U \tag{150}$$

$$z^L \leq z \leq z^U \tag{151}$$

$$q^L \leq q \leq q^U \tag{152}$$

$$t^L \leq t \leq t^U \tag{153}$$

where

$$\begin{aligned} z_j^L &= z_j^U = 0 \quad \forall j \notin J \\ [q_{jk}^L, q_{jk}^U] &= [w_j^L, w_j^U] \times [d_j^L, d_j^U] \quad \forall j,k \\ [t_{jk}^L, t_{jk}^U] &= (e^{j\mathrm{T}} N (N^{\mathrm{T}} N)^{-1} N^{\mathrm{T}} [\Delta x^L, \Delta x^U]) \times (e^{k\mathrm{T}} N (N^{\mathrm{T}} N)^{-1} N^{\mathrm{T}} [\Delta x^L, \Delta x^U]) \end{aligned}$$

Applying the interval LP relaxation to this program gives the end result:

Covering Program

$$\min \quad c^{\mathrm{T}}(\Delta x^{(+)} - \Delta x^{(-)}) \tag{154}$$

$$\text{s.t.} \quad \bar{b}^{i\mathrm{T}}(z^{(+)} - z^{(-)}) + \sum_{j,k} \hat{H}_{jk}^i (t_{jk}^{(+)} - t_{jk}^{(-)} + q_{jk}^{(+)} - q_{jk}^{(-)}) \leq 0 \quad \forall i \in I_A \tag{155}$$

$$\bar{g}_i + \bar{b}^{i\mathrm{T}}(\Delta x^{(+)} - \Delta x^{(-)}) + \sum_{j,k} \hat{H}_{jk}^i (t_{jk}^{(+)} - t_{jk}^{(-)} + q_{jk}^{(+)} - q_{jk}^{(-)}) \leq 0 \quad \forall i \notin I_A \tag{156}$$

$$\underline{v}^{jk\mathrm{T}}\Delta x^{(+)} - \bar{v}^{jk\mathrm{T}}\Delta x^{(-)} - q_{jk}^{(+)} + q_{jk}^{(-)} \leq 0 \qquad \forall j,k \tag{157}$$

$$\bar{v}^{jk\mathrm{T}}\Delta x^{(+)} - \underline{v}^{jk\mathrm{T}}\Delta x^{(-)} - q_{jk}^{(+)} + q_{jk}^{(-)} \geq 0 \qquad \forall j,k \tag{158}$$

$$\underline{h}^{jk\mathrm{T}}z^{(+)} - \bar{h}^{jk\mathrm{T}}z^{(-)} - q_{jk}^{(+)} + q_{jk}^{(-)} \leq 0 \qquad \forall j,k \tag{159}$$

$$\bar{h}^{jk\mathrm{T}}z^{(+)} - \underline{h}^{jk\mathrm{T}}z^{(-)} - q_{jk}^{(+)} + q_{jk}^{(-)} \geq 0 \qquad \forall j,k \tag{160}$$

$$\begin{aligned}\Delta x_j^U(\Delta x_k^{(+)} - \Delta x_k^{(-)}) + \Delta x_k^U(\Delta x_j^{(+)} - \Delta x_j^{(-)}) - \\ \tfrac{1}{2}(q_{jk}^{(+)} - q_{jk}^{(-)} + q_{kj}^{(+)} - q_{kj}^{(-)} + \\ 2(t_{jk}^{(+)} - t_{jk}^{(-)})) \leq \Delta x_j^U \Delta x_k^U\end{aligned} \qquad \forall j,k \tag{161}$$

$$\begin{aligned}\Delta x_j^L(\Delta x_k^{(+)} - \Delta x_k^{(-)}) + \Delta x_k^L(\Delta x_j^{(+)} - \Delta x_j^{(-)}) - \\ \tfrac{1}{2}(q_{jk}^{(+)} - q_{jk}^{(-)} + q_{kj}^{(+)} - q_{kj}^{(-)} + \\ 2(t_{jk}^{(+)} - t_{jk}^{(-)})) \leq \Delta x_j^L \Delta x_k^L\end{aligned} \qquad \forall j,k \tag{162}$$

$$\begin{aligned}\Delta x_j^L(\Delta x_k^{(+)} - \Delta x_k^{(-)}) + \Delta x_k^U(\Delta x_j^{(+)} - \Delta x_j^{(-)}) - \\ \tfrac{1}{2}(q_{jk}^{(+)} - q_{jk}^{(-)} + q_{kj}^{(+)} - q_{kj}^{(-)} + \\ 2(t_{jk}^{(+)} - t_{jk}^{(-)})) \geq \Delta x_j^L \Delta x_k^U\end{aligned} \qquad \forall j,k \tag{163}$$

$$\begin{aligned}\Delta x_j^U(\Delta x_k^{(+)} - \Delta x_k^{(-)}) + \Delta x_k^L(\Delta x_j^{(+)} - \Delta x_j^{(-)}) - \\ \tfrac{1}{2}(q_{jk}^{(+)} - q_{jk}^{(-)} + q_{kj}^{(+)} - q_{kj}^{(-)} + \\ 2(t_{jk}^{(+)} - t_{jk}^{(-)})) \geq \Delta x_j^U \Delta x_k^L\end{aligned} \qquad \forall j,k \tag{164}$$

$$0 \leq \quad 2(t_{jk}^{(+)} - t_{jk}^{(-)}) + |\rho_{jk}^{-1}| \sum_{r,s} \bar{\gamma}_{rs}(t_{rs}^{(+)} - t_{rs}^{(-)}) \qquad \forall j,k \tag{165}$$

$$0 \leq -2(t_{jk}^{(+)} - t_{jk}^{(-)}) + |\tilde{\rho}_{jk}^{-1}| \sum_{r,s} \bar{\gamma}_{rs}(t_{rs}^{(+)} - t_{rs}^{(-)}) \qquad \forall j,k \tag{166}$$

$$z^{(+)} - z^{(-)} = (I + [0 \ N])(\Delta x^{(+)} - \Delta x^{(-)}) \tag{167}$$

$$\Delta x^L \leq \Delta x^{(+)} - \Delta x^{(-)} \leq \Delta x^U \tag{168}$$

$$z^L \leq z^{(+)} - z^{(-)} \leq z^U \tag{169}$$

$$q^L \leq q^{(+)} - q^{(-)} \leq q^U \tag{170}$$

$$t^L \leq t^{(+)} - t^{(-)} \leq t^U \tag{171}$$

$$\bar{b}^{i(+)}_{s(i)} z^{(+)} + \bar{b}^{i(-)}_{s(i)} z^{(-)} - \sum_{j \neq s(i)} (\bar{b}^{i(-)}_j z^{(+)}_j + \bar{b}^{i(+)}_j z^{(-)}_j) - \qquad \forall i \in I_A \tag{172}$$
$$\sum_{j,k} (\hat{H}^{i(-)}_{jk} (t^{(+)}_{jk} + q^{(+)}_{jk}) + \hat{H}^{i(+)}_{jk} (t^{(-)}_{jk} + q^{(-)}_{jk})) \leq 0$$

$$-z^{(-)}_i + (I + [0\ N])^{(-)}_{ii} \Delta x^{(+)}_i + (I + [0\ N])^{(+)}_{ii} \Delta x^{(-)}_i - \qquad \forall i \in J \tag{173}$$
$$\sum_{j \neq i} \Big((I + [0\ N])^{(+)}_{ij} \Delta x^{(+)}_j + (I + [0\ N])^{(-)}_{ij} \Delta x^{(-)}_j \Big) \leq 0$$

$$q^{(-)}_{jk} - \underline{h}^{\mathrm{T}jk(-)} z^{(+)} - \bar{h}^{\mathrm{T}jk(+)} z^{(-)} \leq 0 \qquad \forall j,k \in U \tag{174}$$

$$q^{(+)}_{jk} - \bar{h}^{\mathrm{T}jk(+)} z^{(+)} - \underline{h}^{\mathrm{T}jk(-)} z^{(-)} \leq 0 \qquad \forall j,k \notin U \tag{175}$$

$$(2 + |\rho^{-1}_{jk}|\bar{\gamma}_{jk})^{(-)} t^{(+)}_{jk} + (2 + |\rho^{-1}_{jk}|\bar{\gamma}_{jk})^{(+)} t^{(-)}_{jk} - \qquad \forall (j,k) \in C \tag{176}$$
$$\sum_{r,s \neq j,k} \left(\left(|\rho^{-1}_{jk}|\bar{\gamma}_{rs} \right)^{(+)} t^{(+)}_{rs} + \left(|\rho^{-1}_{jk}|\bar{\gamma}_{rs} \right)^{(-)} t^{(-)}_{rs} \right) \leq 0$$

$$(-2 + |\tilde{\rho}^{-1}_{jk}|\bar{\gamma}_{jk})^{(-)} t^{(+)}_{jk} + (-2 + |\tilde{\rho}^{-1}_{jk}|\bar{\gamma}_{jk})^{(+)} t^{(-)}_{jk} - \qquad \forall (j,k) \notin C \tag{177}$$
$$\sum_{r,s \neq j,k} \left(\left(|\tilde{\rho}^{-1}_{jk}|\bar{\gamma}_{rs} \right)^{(+)} t^{(+)}_{rs} + \left(|\tilde{\rho}^{-1}_{jk}|\bar{\gamma}_{rs} \right)^{(-)} t^{(-)}_{rs} \right) \leq 0$$

where

$$\begin{aligned}
C &= \{(j,k) : |2 + |\rho^{-1}_{jk}|\bar{\gamma}_{jk})| \geq | -2 + |\tilde{\rho}^{-1}_{jk}|\bar{\gamma}_{jk}|\} \\
v^{jk\mathrm{T}} &= \left[d^L_k, d^U_k\right] e^{j\mathrm{T}} (I + N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}}) \\
h^{jk\mathrm{T}} &= \left[w^L_k, w^U_k\right] e^{j\mathrm{T}} (I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}}) \\
U &= \{(j,k) : \sigma_{jk} >= 0\}
\end{aligned}$$

Although large and notationally complicated, this program is a straight forward translation of (138). As the interval relaxation dictates, all of the variables get replaced by their positive and negative components, and each original constraint is transformed into the new program. Constraints (155) and (156) correspond to constraints (139) and (140) in the interval orthant program, and likewise constraints (157–160) correspond to constraints (141) and (142). Each of the constraints (161–171) corresponds to a constraint in (138).

The interval relaxation also includes a constraint for each variable of the form (101). For each variable, a particular constraint is chosen to make an extra constraint to tighten the bound. Constraints (172–177) are these extra constraints chosen for variables. Constraint (172) is the constraint for the z variables, and $s(i)$ is defined by the pivot sequence used in LU factoring G. The x constraints (173) are taken from the equation (149) that relates x and z and the q_{jk} constraints (174) and (175) are taken from constraint (142) using σ_{jk} to indicate whether the upper or lower bound will be active. Lastly, the t constraints (176) and (177) are chosen from (147) and (148) using the constraint with the largest coefficient on t_{jk}.

When there is no null space, the interval LP can be simplified to the following:

$$\min \quad c^{\mathrm{T}}\Delta x \tag{178}$$

$$\text{s.t.} \quad \bar{g}_i + \bar{b}^{i\mathrm{T}}\Delta x + \sum_{j,k} \hat{H}^{(i)}_{jk} q_{jk} \leq 0 \qquad \forall i \tag{179}$$

$$0 \leq [\Delta x^L_k, \Delta x^U_k]\Delta x_j - q_{jk} \leq 0 \qquad \forall j,k \tag{180}$$

$$\Delta x^U_j \Delta x_k + \Delta x^U_k \Delta x_j - q_{jk} \leq \Delta x^U_j \Delta x^U_k \quad \forall j,k \tag{181}$$

$$\Delta x^L_j \Delta x_k + \Delta x^L_k \Delta x_j - q_{jk} \leq \Delta x^L_j \Delta x^L_k \quad \forall j,k \tag{182}$$

$$\Delta x^L_j \Delta x_k + \Delta x^U_k \Delta x_j - q_{jk} \geq \Delta x^L_j \Delta x^U_k \quad \forall j,k \tag{183}$$

$$\Delta x^U_j \Delta x_k + \Delta x^L_k \Delta x_j - q_{jk} \geq \Delta x^U_j \Delta x^L_k \quad \forall j,k \tag{184}$$

$$\Delta x^L \leq \Delta x \leq \Delta x^U \tag{185}$$

$$q^L \leq q \leq q^U \tag{186}$$

where

$$[q_{jk}^L, q_{jk}^U] = [\Delta x_j^L, \Delta x_j^U] \times [\Delta x_k^L, \Delta x_k^U]$$

Applying the interval LP relaxation gives the following useful result:

$$\begin{aligned}
\min \quad & c^{\mathrm{T}}(\Delta x^{(+)} - \Delta x^{(-)}) && && (187)\\
\text{s.t.} \quad & \bar{g}_i + \bar{b}^{i\mathrm{T}}(\Delta x^{(+)} - \Delta x^{(-)}) + \sum_{j,k} \hat{H}_{jk}^i (q_{jk}^{(+)} - q_{jk}^{(-)}) \le 0 && \forall i && (188)\\
& \Delta x_k^L \Delta x_j^{(+)} - \Delta x_k^U \Delta x_j^{(-)} - q_{jk}^{(+)} + q_{jk}^{(-)} \le 0 && \forall j,k && (189)\\
& \Delta x_k^U \Delta x_j^{(+)} - \Delta x_k^L \Delta x_j^{(-)} - q_{jk}^{(+)} + q_{jk}^{(-)} \ge 0 && \forall j,k && (190)\\
& \Delta x_j^U (\Delta x_k^{(+)} - \Delta x_k^{(-)}) + \Delta x_k^U (\Delta x_j^{(+)} - \Delta x_j^{(-)}) - && \forall j,k && (191)\\
& \qquad q_{jk}^{(+)} + q_{jk}^{(-)} \le \Delta x_j^U \Delta x_k^U && && \\
& \Delta x_j^L (\Delta x_k^{(+)} - \Delta x_k^{(-)}) + \Delta x_k^L (\Delta x_j^{(+)} - \Delta x_j^{(-)}) - && \forall j,k && (192)\\
& \qquad q_{jk}^{(+)} + q_{jk}^{(-)} \le \Delta x_j^L \Delta x_k^L && && \\
& \Delta x_j^L (\Delta x_k^{(+)} - \Delta x_k^{(-)}) + \Delta x_k^U (\Delta x_j^{(+)} - \Delta x_j^{(-)}) - && \forall j,k && (193)\\
& \qquad q_{jk}^{(+)} + q_{jk}^{(-)} \ge \Delta x_j^L \Delta x_k^U && && \\
& \Delta x_j^U (\Delta x_k^{(+)} - \Delta x_k^{(-)}) + \Delta x_k^L (\Delta x_j^{(+)} - \Delta x_j^{(-)}) - && \forall j,k && (194)\\
& \qquad q_{jk}^{(+)} + q_{jk}^{(-)} \ge \Delta x_j^U \Delta x_k^L && && \\
& \Delta x^L \le \Delta x^{(+)} - \Delta x^{(-)} \le \Delta x^U && && (195)\\
& q^L \le q^{(+)} - q^{(-)} \le q^U && && (196)\\
& \quad \bar{b}_{s(i)}^{i(+)} \Delta x^{(+)} + \bar{b}_{s(i)}^{i(-)} \Delta x^{(-)} - && \forall i && (197)\\
& \sum_{j \ne s(i)} (\bar{b}_j^{i(-)} \Delta x_j^{(+)} + \bar{b}_j^{i(+)} \Delta x_j^{(-)}) - && && \\
& \sum_{j,k} (\hat{H}_{jk}^{i(-)} q_{jk}^{(+)} + \hat{H}_{jk}^{i(+)} q_{jk}^{(-)}) \le 0 && && \\
& q_{jk}^{(-)} - \Delta x_k^{L(-)} \Delta x_j^{(+)} - \Delta x_k^{U(+)} \Delta x_j^{(-)} \le 0 && \forall j,k \in U && (198)
\end{aligned}$$

$$q_{jk}^{(+)} - \Delta x_k^{U(+)} \Delta x_j^{(+)} - \Delta x_k^{L(-)} \Delta x_j^{(-)} \le 0 \qquad \forall j, k \notin U \tag{199}$$

$$\tag{200}$$

The above give two forms of the covering program available to calculate a lower bound for the NLP over a range of variables. LP (187) is much smaller and is used when the problem is completely constrained, while the larger (154) is used when the problem has a null space. This decision is based on the results from a previous solution. In addition, the solution to these gives a step for the x variables, Δx, which will be used in a line search in the algorithm.

4.6 Adding a Newton Constraint

One problematic feature of the null space covering program is that there is no constraint relating t_{jk} to Δx except for McCormick's envelopes. When the program can reduce its objective function by moving away from $\check{x}$ in the direction of the null space, McCormick's constraints and variable bounds are the only constraints that keep it from moving infinitely in the null space direction, and these constraints do not give a Δx which will lead the iterative algorithm to the global minimum.

In this section, a constraint which can be added to the covering program to limit motion in the null space will be developed. It is designed to produce a Newton step in the null space.

Assuming that the active set is correct, the LU factors of G used in calculating

N may also be used to calculate the Newton step in the constraint space.

$$d^N = -\begin{bmatrix} G_I^{-T} \\ 0 \end{bmatrix} \bar{g} \tag{201}$$

The KKT conditions of the quadratic subproblem in successive quadratic programming are:

$$\begin{aligned} \nabla f(x_k) + \nabla^2 L(x_k)\Delta x + \sum_{i=1}^{m} u_i \nabla g_i(x_k) &= 0 &\quad (202)\\ u_i\left(g_i(x_k) + \nabla g_i(x_k)^{\mathrm{T}}\Delta x\right) &= 0 &\quad (203)\\ u &\geq 0 &\quad (204)\end{aligned}$$

By premultiplying by N^{T}, the contributions of the active constraints in the first equality are eliminated.

$$N^{\mathrm{T}}\nabla f(x_k) + N^{\mathrm{T}}\nabla^2 L(x_k)\Delta x = N^{\mathrm{T}}\nabla f(x_k) + N^{\mathrm{T}}\gamma\Delta x = 0 \tag{205}$$

If x is broken into its constraint space and null space parts, the following requirement for a Newton step in the null space are obtained.

$$N^{\mathrm{T}}\gamma N p^N = Q p^N = -N^{\mathrm{T}}(\nabla f(x_k) + \gamma d^N) \tag{206}$$

In constructing the null space quadratic term bounds, the Cholesky factors of Q are calculated, so it costs little to use equation (206) to calculate the Newton step in the null space. The constraint space and null space Newton steps can be added to give the complete Newton step which will be used in the line search in the algorithm.

$$\begin{aligned} p^N &= -Q^{-1}N^{\mathrm{T}}(\nabla f(x_k) + \gamma d^N) &\quad (207)\\ \Delta x^N &= d^N + N p^N &\quad (208)\end{aligned}$$

By assumption Q is a symmetric, positive semi-definite matrix, so

$$p^{\mathrm{T}} Q p \geq 0$$

for all p. Choosing the correct values of p will produce a constraint which is valid and can give a Newton step in the null space direction.

$$(p-\hat{p})^{\mathrm{T}} Q (p-\hat{p}) \geq 0 \tag{209}$$

$$\overbrace{p^{\mathrm{T}} Q p}^{\sum_{ij} \gamma_{ij} t_{ij}} -2\hat{p}^{\mathrm{T}} Q p + \hat{p}^{\mathrm{T}} Q \hat{p} \geq 0 \tag{210}$$

$$\sum_{ij} \gamma_{ij} t_{ij} \geq 0 \tag{211}$$

The goal is to choose a set of $\hat{p}$'s such that when constraints (210) and (211) are active, the Newton step in the null space given by (206), so (210) and (211) will be treated like equalities below. By scaling each row of (206) by $-\alpha_m$, the value of $\hat{p}$ which makes (210) equivalent to a row of (206) when (211) is active can be determined.

$$-\alpha_m e^{m\mathrm{T}} Q p = \alpha_m e^{m\mathrm{T}} N^{\mathrm{T}} (\nabla f(x_k) + \gamma d^N) \tag{212}$$

$$-2\hat{p}^{\mathrm{T}} Q p = -\hat{p}^{\mathrm{T}} Q \hat{p} \tag{213}$$

$$\hat{p} = \frac{\alpha_m}{2} e^m \tag{214}$$

$$\alpha_m = \frac{-4 e^{m\mathrm{T}} N^{\mathrm{T}} (\nabla f(x_k) + \gamma d^N)}{Q_{mm}} \tag{215}$$

These values of α and $\hat{p}$ give a constraint of the following form for each null space dimension.

$$\sum_{ij} \gamma_{ij} t_{ij} - \alpha_m e^{m\mathrm{T}} Q p \geq \alpha_m e^{m\mathrm{T}} N^{\mathrm{T}} (\nabla f(x_k) + \gamma d^N) \tag{216}$$

The combination of constraints (216) and (147–148) constrain the LP's movement in the null space.

4.7 Program Size

Bounding the original NLP requires additional variables and constraints, resulting in a covering program that is a larger LP. In this section, the size of the covering program including the Newton constraints is related to the problem characteristics.

Given a problem in factorable form (11), the following index sets are defined:

$$\bar{K} = \{(j,k) \in 1 \dots n \times 1 \dots n | \exists i \in 1 \dots m, (H_{jk}^{(i)} \neq 0 \vee j = k, j \in F_i)\} \quad (217)$$

$$\bar{T} = \{(j,k) \in K | j \leq k\} \quad (218)$$

$\bar{K}$ is the index set required for q_{jk}, and $\bar{T}$ is the initial index set required for t_{jk}. The cardinality of a set is denoted with the notation $|\cdot|$. The number of active constraints is $|I_A| + |I'_A|$, and the null space has dimension $n - (|I_A| + |I'_A|)$. In section 3.4, some additional quadratic terms may be added to adjust the Q matrix; let a be the number of quadratic terms added in this step. It can be shown that $a \leq n - (|I_A| + |I'_A|)$.

For the first form of the covering program (154), the number of variables and rows can be calculated and bounded as follows:

$$\begin{aligned}
\text{Number Variables} &= 2(n + |I_A| + |I'_A| + |\bar{K}| + |\bar{T}| + a) \\
\text{Number Variables} &\leq 2(2n + |\bar{K}| + |\bar{T}|) \\
\text{Number Rows} &= m + 2n + |I_A| + |I'_A| + 9|\bar{K}| + 3(|\bar{T}| + a) \\
\text{Number Rows} &\leq m + 5n + 9|\bar{K}| + 3|\bar{T}|
\end{aligned}$$

The second form of the covering program (187) always has the same number of variables and rows because it does not depend on the null space. The number of

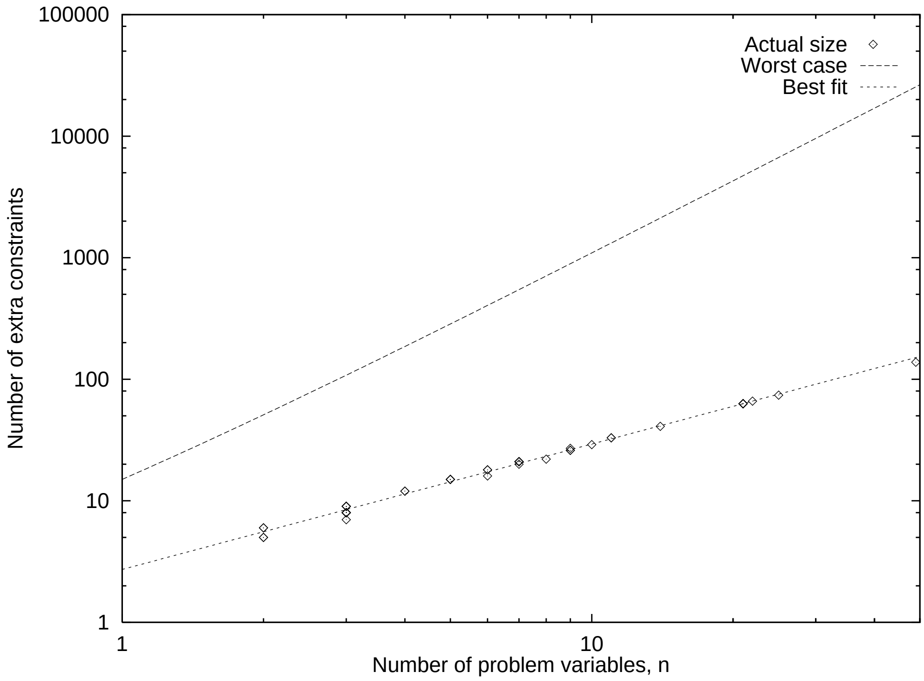


Figure 7: Number of extra constraints versus number of problem variables

variables and rows required is given by

$$\begin{aligned} \text{Number Variables} &= 2(n + |\bar{K}|) \\ \text{Number Rows} &= m + n + 7|\bar{K}|. \end{aligned}$$

A completely dense problem presents the worst case for covering program sizes giving $|\bar{K}| = n^2$ and $|\bar{T}| = \frac{1}{2}n(n+1)$. Figure 7 shows a comparison of the number of extra constraints (in addition to the m original constraints) required in the null space version of the covering program applied to the test problems used in the next chapter and the worst case behavior. The slope for the line of best fit through the test points is 1.03, indicating linear growth in the number of extra constraints with respect to the number of original problem variables.

Chapter 5

The Algorithm

This algorithm falls into the class of branch and bound algorithms for solving NLPs discussed in Chapter 2. It uses the covering program to calculate a lower bound for the NLP and uses the objective function value at feasible points as an upper bound on the solution of the problem. When the lower bound for the NLP equals the upper bound, the problem is solved. If the algorithm is applied to a problem without a solution, the algorithm determines this by eliminating all subsets of the variable domain because of infeasibility. The overall goal during the algorithm is to increase the lower bound and to decrease the upper bound until they meet.

In difficult problems, the covering program applied to the original domain will provide a lower bound significantly lower than the global optimum, so the branch and bound algorithm must resort to an adaptive domain partitioning strategy to increase the lower bound. The original variable domain is split into a list of subsets which are bounded separately. The lower bound for the NLP is the least lower bound of all of the regions in the list. Any member of the list can be removed if its lower bound is greater than or equal to the current upper bound or if its covering program is infeasible. The algorithm proceeds by bounding the region with the least lower bound, and if it cannot be ruled out, it is removed from the list and split into two subsets which are added to the list. As the region size decreases, the lower bound increases until all of the regions can be ruled out and the global optimum is verified or all of the regions are infeasible.

Decreases in the upper bound are obtained in the course of searching for better feasible points. The program stores the best feasible point and its objective function value. Each time the problem constraints are evaluated at a point, the program checks to see if it is a better feasible point than the stored one, and if it is, it replaces the stored one. The systematic search for better feasible points is accomplished by the region analysis procedure. Two different versions of the region analysis procedure have been developed for comparison. The first is a modification of the algorithm presented by Swaney [114] and uses MINOS 5.4 [74] to search for local minima of the NLP. The second uses a line search in the direction provided by the covering program solution, and the direction provided by a Newton step if available, to search for new feasible points.

The algorithm presentation below will be organized in three main parts: the branch and bound loop, the variable splitting method, and the region analysis procedure.

5.1 The Branch and Bound Loop

The branch and bound loop is the manager for the algorithm. It organizes and maintains the region list, and it makes the choice of which region to analyze next. It keeps track of the current upper and lower bounds, and prunes the region list when needed. All of the other procedures operate under its control.

One of the primary functions of the branch and bound loop is to manage the region list. Each element in the region list contains the specification of a region describing the lower and upper bounds for each variable and a lower bound on the objective function within that region. This list is kept sorted in order of increasing lower bounds, and the next region to be analyzed is always taken from the top of the list. This sorting and region selection corresponds to analyzing the region with the lowest lower bound first. The algorithm has also been operated in a highest lower bound first manner, in a last in first out (LIFO) manner, and first in first out (FIFO) manner. However, the lowest lower bound appears to be the best based on tests conducted early in the algorithm development.

The region list is initialized with the variable bounds from the problem specification, and the best objective function value is initialized to a large value. The algorithm terminates when the region list is empty.

The procedures that evaluate the objective function and constraints keep track of the best feasible point. Each time the constraints are evaluated, the procedure checks if the point is feasible (not violating any of the constraints by more than $\epsilon^{0.55} = 2.4578 \times 10^{-9}$) and if it is better than the current best point. The evaluation procedures also store the gradient of the constraints at the best feasible point to avoid recalculating them. The branch and bound loop can poll the evaluation routines to obtain the best objective function value, the best point, and the number of feasible points found.

The objective function value of the best feasible point can be used to remove elements from the region list. Any region list element with a lower bound greater than or equal to the current best objective function value cannot contain a better feasible point, so it is removed from the region list (pruned).

Here is a summary of the algorithm:

Branch and bound loop

Initialize search list

While Search list not empty

1. Remove the top of the search list to be treated as
   the current region.
2. Choose a point inside the current region using the current
   point or the midpoint of the region
3. Analyze the current region (using region analysis procedure)
4. Prune region list if a better feasible point was found
5. If the current region is not pruned due to bound or
   infeasibility, choose a split variable and a split
   location and add the two new regions to the region list.

These steps are described further below.

5.1.1 Step 2: Point Choosing Scheme

The covering program requires a point around which to construct its bounds. In this step, the algorithm first checks if the best feasible point is contained in the region of interest; if so, it chooses the best feasible point. Otherwise, the algorithm adapts the point used in the previous iteration (the "current point"). The algorithm checks if each component x_i of the current point is inside the bounds for that variable in the current region. If it is inside the bounds, its value remains unchanged; otherwise, x_i is assigned the value of the average of the lower and upper bounds for x_i. If the current point is inside the region, region analysis can be started without having to reevaluate the objective function, constraints, and the constraint gradients.

5.1.2 Step 4: Prune the Region List

When a new feasible point is found, the algorithm checks each element in the region list to see if it can be pruned. Any region whose bound is greater than the current upper bound can be removed from the list. This can reduce the amount of work and storage required. For finite precision mathematics, a region is pruned if (upper bound − lower bound) $\leq 1 \times 10^{-4}$. The problems are scaled using a heuristic method described below which attempts to give the objective function an order of magnitude of 10^0. This pruning criteria guarantees the objective function to a high enough tolerance for most engineering applications. However, problems with many local minima with objective function values very close to each other may require a higher tolerance.

5.2 Step 5: Splitting the Region

The method for choosing which variable to split and where it should be split has a large effect on the performance of the overall algorithm. Each time the algorithm splits it creates two more regions that may need to be stored and processed or pruned, so it is important that each split be chosen judiciously. Otherwise, the algorithm may just generate more work without improving the lower bound. The goal of the method presented here is to choose the variable which will most greatly improve the objective function bound.

The method used is analogous to the method suggested by Swaney [114]. The starting idea is to choose the variable whose bounds most greatly affect the value of the covering program objective function. The effect on the objective function of changes in the bounds is estimated from a sensitivity analysis of the solution to the covering program. Given the following program which depends on certain parameters a

$$\begin{aligned} \min_x \quad & \varphi(x) \\ \text{s.t.} \quad & g(x,a) \le 0 \end{aligned}$$

and the solution $\varphi^* = \varphi(x^*)$, the effect of a change in the parameters, Δa, can be estimated as follows:

$$\Delta\varphi^* \approx u^{*\mathrm{T}} \left.\frac{\partial g}{\partial a^{\mathrm{T}}}\right|_{x=x^*} \Delta a \tag{219}$$

where u^* are the Lagrange multipliers at the solution. For the variable splitting procedure, the parameters are the variable bounds and the program is either (154) or (187). For the null space program, constraints (155–156) are affected by changes in variable bounds through changes in $\hat{H}^{(i)}$, (157–164), (168–171), and

(174–175), and similarly for the full rank program, constraints (188–196) and (198–199) are affected. To improve efficiency, $\left.\frac{\partial g}{\partial a^{\mathrm{T}}}\right|_{x=x^*}$ is only determined for the active constraints, so the effect of a bound change may be estimated with reasonable effort.

The process of splitting takes place in two parts. First the variable to split is chosen, and then the actual split location is chosen. In the first step, it is necessary to choose a hypothetical split location. For each variable, a point in its domain, x_i^c, is chosen and the effects of changing the bounds from $[x_i^L, x_i^U]$ to $[x_i^L, x_i^c]$ and $[x_i^c, x_i^U]$ are estimated. x_i^c is chosen in the following manner:

$$x_i^c = \begin{cases} \check{x}_i & \text{if } x_i^L < \check{x}_i < x_i^U \\ \check{x}_i + \Delta x_i & \text{otherwise if } x_i^L < \check{x}_i + \Delta x_i < x_i^U \\ \frac{1}{2}(x_i^L + x_i^U) & \text{if neither of the above is satisfied} \end{cases} \tag{220}$$

where Δx_i comes from the solution from the appropriate covering program.

The effects of changing each variable's lower and upper bound are estimated and labeled Δ_i^L and Δ_i^U respectively. For purposes of choosing the variable to split, it is better to reduce the contribution of the simple variable bounds on the x and z variables, constraints (168–169) or (195–196), to a small percentage of their contribution. The goal of splitting is to improve the bounds on the quadratic terms, and the effect of simple variable bounds should be secondary. The results presented in the following chapter are for an algorithm using 1% of the contribution from the simple variable bounds. The estimates with the reduced contribution from the simple variable bounds are called $\Delta_i^{L\prime}$ and $\Delta_i^{U\prime}$. The variable to split on is chosen by the following:

$$l = \arg\max_i \tfrac{1}{2}(\Delta_i^{L\prime} + \Delta_i^{U\prime}) \tag{221}$$

If Δ_l^L and Δ_l^U should be zero, the nonlinear variable with the widest domain is bisected, and Step 5 ends causing the algorithm to continue with Step 1.

When Δ_l^L and Δ_l^U are not zero, the algorithm now switches to the task of choosing the split location. If an upper bound for the problem is available, the algorithm attempts to determine a split which will result in one region that can be eliminated by a lower bound increase and one region that still might contain the global minimum. This procedure may fail if the estimated bound increase is not large enough to predict that part of the region may be eliminated or if it requires a division by a number near zero. If this procedure should fail or if an upper bound is not available, the algorithm will split at x_l^c as long as it satisfies the following conditions

$$x_l^L + 0.2(x_l^U - x_l^L) \le x_l^c \le x_l^U - 0.2(x_l^U - x_l^L)$$

Otherwise, the variable is split at $\frac{1}{2}(x_l^L + x_l^U)$.

When an upper bound is available, the estimated increase of the covering program objective function can be used to determine a split location that will yield one region that can be eliminated and another than may contain the solution. The lower bound given by the covering program solution is $\bar{x}_0 + \Delta x_0$, and the estimated new bound as a function of split location, x_l^m, can be written as

$$\begin{aligned} x_0^{\text{est}} &= \bar{x}_0 + \Delta x_0 + \frac{\Delta_l^L}{x_l^c - x_l^L}(x_l^m - x_l^L) \\ x_0^{\text{est}} &= \bar{x}_0 + \Delta x_0 + \frac{\Delta_l^U}{x_l^U - x_l^c}(x_l^U - x_l^m) \end{aligned}$$

The goal is for x_0^{est} to meet or exceed the upper bound, UB, so by substituting in for $x_0^{\rm est}$, the following equations for x_l^m are obtained.

$$x_l^m = \phi \frac{UB - (\bar{x}_0 + \Delta x_0)}{\Delta_l^L}(x_l^c - x_l^L) + x_l^L, \quad (222)$$

$$x_l^m = x_l^U - \phi \frac{UB - (\bar{x}_0 + \Delta x_0)}{\Delta_l^U}(x_l^U - x_l^c) \quad (223)$$

where ϕ is a small multiplier ($1 \le \phi \le 1.2$) to overestimate the cut needed. If the estimate for one of the bound changes lies outside the variable domain or if Δ_l is zero, this analysis for that bound cannot predict a split that will eliminate part of the region. The splitting algorithm chooses that value of x_l^m which will eliminate the largest piece of the region. For a lower bound change, $x_l^U - x_l^m$ is predicted to be eliminated, and for an upper bound change, $x_l^m - x_l^L$ is predicted to be eliminated.

After the variable and location have been recommended, the algorithm checks the ratio of the width of the recommended variable to the width of the widest nonlinear variable normalized by their original widths. If the recommended variable's normalized width is less than one one-hundredth of the widest nonlinear variable's normalized width, the algorithm overrides the recommendation and bisects the widest nonlinear variable.

After the splitting method has chosen a variable and a split location, it needs to add the two new regions to the branch and bound list. The new regions take their lower bound on the objective function from the region they subdivided because their bounds lower have to be at least as high as the region from which they come. If their bound is the same as the first element in the list, they are added to the front of the list; otherwise, they are inserted in order of increasing lower bound. The region which includes the current point (the point around which the covering program was constructed) is added second (on top). If this region has the least lower bound, the covering program of the next iteration can be constructed without having to evaluate the objective function, constraints, and their gradients again. This is not necessary for the algorithm to work, but reduces the number of function evaluations.

5.3 The Region Analysis Procedure, Version 1

This region analysis procedure is a modification of the one presented by Swaney [114]. It is different because it uses MINOS 5.4 [74] as its local NLP solver and because it uses the generalized covering program.

Region analysis procedure, version 1

1. Call MINOS 5.4 to solve the NLP (11)
   restricted to $[x^L, x^U]$ using the point provided
   by the branch and bound loop as a starting place.
2. Construct the covering problem at the point found by
   MINOS 5.4, and return the results to the branch and bound
   loop.

Because the generalized covering program is used, this region analysis loop can still provide a lower bound even when MINOS 5.4 does not find a feasible point. If the problem is infeasible, the covering program will become infeasible when the region size is small enough. The details of how to construct the covering program are shown in the next section.

5.4 The Region Analysis Procedure, Version 2

The second version of the region analysis procedure is a new approach combining a search direction derived from the covering program and a Newton like search direction. The covering program provides a search direction based on the global character of the problem which may provide an improved point when the local methods fail. The local step is used to provide quick convergence when close to a local minimum.

The region analysis procedure is designed to find a global minimizer of a particular region and to provide a lower bound on the optimal objective function value in the region. It is also desirable for the loop to find feasible points as soon as possible.

Region analysis procedure, version 2

Loop

1. Construct and solve the covering program at the current point.
2. If the covering program is infeasible or if the lower bound allows this region to be pruned, exit the loop.
3. Update the region's lower bound estimate from covering program.
4. If the covering program was stationary ($||\Delta x|| = 0$)

   If the current point is feasible, exit the loop (optimum found).

   Else, exit because zero search direction.
5. Perform a line search.
6. Check if the current region can be pruned.

5.4.1 Step 1: Constructing the Covering Program

To construct the appropriate covering program, a point and an estimate of the Lagrange multipliers at that point are needed. If an estimate of the Lagrange multipliers is not available (as on the first iteration), the interval LP with no null space (187) can be used. After solving each covering program, the algorithm stores the Lagrange multipliers for use by the next covering program. When working with the null space covering program, the Lagrange multiplier estimates for the general constraints come from the LP Lagrange multipliers from constraints (155–156) and (172). When working with the full rank covering program, the Lagrange multiplier estimates come from (188) and (197).

Given Lagrange multiplier estimates, the algorithm first counts the number of active constraints as determined by the value of the multipliers and MINOS' basis array. If the number of active constraints is less than the number of variables, the algorithm constructs the null space covering program; otherwise, it constructs the full rank covering program.

Constructing the full rank program (187) is relatively straightforward. Given the current point and the bounds on the variables, constructing the LP does not require any difficult operations.

Constructing the null space program (154) requires more computational effort. First the algorithm constructs the matrix G, defined by (22), whose columns are gradients of the active constraints determined from the Lagrange multipliers. Next the LU factors of G are calculated using LUSOL [30], the sparse LU factorization routine from MINOS 5.4 [74]. If G is not of full rank, the linearly dependent constraints are removed from the active set, and G is recalculated until a set of linearly independent active constraint gradients is found.

Once a G of full rank has been found, the algorithm uses it to calculate new estimates of the multipliers. Assuming that the current point is a Karush-Kuhn-Tucker point,

$$c + Gu = 0$$

which is solved for u using the LU factors of G. These updated u estimates are checked to make sure they are of the correct sign. If there is a sign error, the associated constraint is removed from the active set, and a new G is used.

Using the LU factors of G, a basis for the null space of the active constraints, N, is calculated.

$$N = \begin{bmatrix} G_I^{-\mathrm{T}} G_{II}^{\mathrm{T}} \\ -I \end{bmatrix} \tag{23}$$

where G_I are the most linearly independent rows of G as determined by row permutation provided by the LU factorization routines. The Newton step in the constraint space, d^N, is computed from the LU factors of G using (201).

Once N has been calculated, the projection matrix, $N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}}$, is calculated using the QR factorization of N, from which $N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}} = Q_N Q_N^{\mathrm{T}}$. This matrix is calculated and stored using full matrix routines.

Next γ is calculated as $\sum_i u_i \bar{H}^{(i)}$. Matrix Q is calculated from γ according to the definition (46). Q is stored and processed as a full matrix. Q is factorized using a modified Cholesky factorization enforcing positive definiteness [107] which adds the smallest possible diagonal matrix needed to Q to make it positive definite. That procedure was modified to add a larger diagonal matrix to avoid scaling problems in the LP. The diagonal adjustment is propagated to γ using (54). From Q, η is calculated using full matrix computations using equation (50), and from η, the required elements of ρ^{-1} and $\tilde{\rho}^{-1}$ are calculated using (50). The Cholesky factors of Q are also used to calculate the Newton step in the null space, p^N, using (207) which gives the total Newton step of $\Delta x^N = d^N + Np^N$. The α's for the Newton constraints are calculated using equation (215).

From this information, the null space covering program can be constructed and solved using MINOS 5.4.

5.4.2 Step 5: The Line Search

The algorithm requires a special kind of line search that can search two directions simultaneously. In some cases, the algorithm will have a search direction from the covering program, Δx^{C}, and a Newton search direction, Δx^{N}, and sometimes, it will have only one of the two. When the full rank bound is used, only Δx^{C} is available, and when the covering program gives a zero step or cannot be solved by MINOS 5.4 (which occurs very rarely), only the Newton step is available.

The line search uses the same merit function for evaluating progress for either search direction. The merit function contains the objective function and a weighted sum of the infeasibility.

$$P(x) = f(x) + \sum_i W_i \max\{g_i(x), 0\} \tag{224}$$

The weighting factors W are a kind of moving average of the Lagrange multipliers.

W^{k+1} is calculated from W^k using the first rule that applies in the following:

$$W_i^{k+1} = \begin{cases} W_i^k & \text{if } u_i = 0 \\ 2|u_i| & \text{if } W_i^k = 0 \\ \frac{1}{2}(W_i^k + 2|u_i|) & \text{otherwise} \end{cases} \tag{225}$$

This definition allows the weights to be adjusted without drastic changes that can cause cycling.

The line search first calculates the directional derivative for each available search direction. If the Newton directional derivative is nonnegative which may occur if the active set is wrong, the Newton step is ignored. When the directional derivative is negative, an Armijo criteria [10, section 8.3] is used as a terminating condition; otherwise, a fixed decrease is required for the step be accepted. The line search takes a step with all available directions, and then pursues the one that has the best merit function.

After taking the first step, if the directional derivative is negative, the minimum of a quadratic approximation of the merit function is used to calculate a new step length. If the directional derivative is positive, it decreases the step by 0.4 each iteration. The line search continues until an improved point is found or until a maximum of 8 iterations have been taken.

5.5 Problem Scaling

As with many numerical algorithms, this algorithm can fail or perform poorly if the problems are not well scaled. The termination criteria and matrix pivot elements are chosen using absolute tolerances, so it is important that the variables, objective function value, and constraint values be of approximately the same order of magnitude from problem to problem. It is necessary to develop a method to scale the variable values, objective function value, and constraint values.

This algorithm uses a simple, *a priori* procedure to scale the variables and function values. After a problem has been defined as published, the scaling procedure is applied to compute variable and function scales, which are added to the problem definition file before executing the branch and bound algorithm.

The variable scale is based on the order of magnitude of the average of the upper and lower bound. Here are the definitions of some of the functions used and how the scale for variable i is determined.

$$\mathrm{trunc}(y) = \begin{cases} \max\{i \in \text{Integers} \mid i \leq y\} & \text{if } y \geq 0 \\ \min\{i \in \text{Integers} \mid i \geq y\} & \text{if } y < 0 \end{cases} \tag{226}$$

$$\mathrm{Magnitude}(y) = \begin{cases} 10^{\mathrm{trunc}(\log_{10}(|y|))} & \text{if } y \neq 0 \\ 1 & \text{if } y = 0 \end{cases} \tag{227}$$

$$\mathrm{VarScale}(i) = \mathrm{Magnitude}(\frac{\underline{x}_i + \bar{x}_i}{2}) \tag{228}$$

To determine the constraint scale, 100 points are chosen at random uniformly distributed in the variable domain determined by the upper and lower bounds, and the constraint magnitudes are estimated from the average of the largest magnitude term in each expression as explained below. The random points are designated $\dot{x}^k$ for k ranging from 1 to 100. It is assumed that problems are in the form (11) with one possible modification. If the objective function of the original problem specification is linear, the linear objective function is used as published, and x_0 is not introduced as a dummy objective variable.

$$\text{ObjectiveMax}(\dot{x}) = \max_{j \in 1 \ldots n} |c_j \dot{x}_j| \tag{229}$$

$$\text{ObjectiveScale} = \text{Magnitude}(\frac{1}{100} \sum_{k=1}^{100} \text{ObjectiveMax}(\dot{x}^k)) \tag{230}$$

Maximums and absolute values give the scale of the largest term in the expression. The method for the constraints is similar.

$$\text{ConstMax}(i, \dot{x}) = \max\{\max_{j \in 1 \ldots n} |b_j^i \dot{x}_j|, \max_{j,k \in 1 \ldots n} |\dot{x}_j H_{jk}^{(i)} \dot{x}_k|, \max_{j \in F_i} |f_j^i(\dot{x}_j)|\} \tag{231}$$

$$\text{ConstScale}(i) = \text{Magnitude}(\frac{1}{100} \sum_{k=1}^{100} \text{ConstMax}(i, \dot{x}^k)) \tag{232}$$

5.6 Nondifferentiable or Undefined Functions

Some of the single variable functions or their derivatives are undefined for some subset of the variable domain, and to solve them with this algorithm, small adjustments must be made to the problem definition. Of all of the single variable functions needed to define the 50 test problems, $p_0 x^{p_1}$, $p_0 \ln(p_1 x + p_2)$, and $p_0 x \ln(p_1 x + p_2)$ are the only three that may have undefined function values or derivatives. With $\ln(\beta), \beta \leq 0$, undefined, the bounds on variable x may need to be adjusted to keep the lower bound of $p_1 x + p_2$ greater than or equal to some small value strictly greater than zero.

The power function $p_0 x^{p_1}$ is a more complicated problem. The restrictions on x depend on the value of p_1. There are different conditions for p_1 being positive or negative, integral or real, and greater than or less than one. The case that is actually required in the test problems has $p_1 = 0.6$, resulting from the estimated cost of heat exchangers taken as proportional to $(\text{Area})^{0.6}$. In this case, the power function is defined for all nonnegative values of x, but the derivative is undefined at zero. The derivative approaches infinity as x approaches zero from the right. To treat this infinite derivative, the power function is replaced by a fourth order polynomial which goes smoothly to zero for $x <= 0.05$. The fourth order polynomial chosen has the minimum integrated error squared such that the value and derivative of the power function are matched at $x = 0.05$ and the value of power function is matched at $x = 0$.

Chapter 6

The Parallel Algorithm

One of the frequently noted characteristics of branch and bound algorithms is that they involve the solution of numerous independent problems which can be analyzed by separate processors in a parallel scheme. There are two major approaches to parallelize branch and bound algorithms. The first applies all of the processors in parallel to solving one subproblem (method A), and the second assigns each processor to a different subproblem (method B) to solve several subproblems simultaneously. Method A has been taken by a number of researchers in global optimization [79, 87, 88, 89] who report favorable speedups. They achieve parallelism through the parallel solution of multiple-cost-row programs, which has finer granularity than method B. Other researchers have studied parallelism at the subproblem level [50, 61, 68, 104], and they have reported results ranging from superlinear speedup (speedup anomaly) to situations where adding more processors causes the algorithm take longer (detrimental anomaly).

The parallel version of the algorithm presented here uses method B for a number of reasons: lack of parallel numerical libraries, ease of implementation, and suitability for different parallel environments. Applying method A to this parallel branch and bound algorithm would require parallel sparse LP and NLP libraries which were not readily available. Adapting the sequential algorithm to method B only required small additions to the source code and was completed in three weeks. Method B has a very low communication load in comparison to method A, which would require more messages to be passed, so method B will be more competitive on parallel machines lacking high speed communications capabilities such as networks of workstations. For the remainder of the chapter, method B will be considered for a parallel computer with p processors (sometimes referred to as nodes.)

The anomalies that occur with method B depend on the number of regions that the algorithm has to examine. The speedup anomaly occurs when the parallel branch and bound algorithm examines fewer regions than the sequential version; this can occur when the global optimum is found earlier allowing some regions evaluated in the sequential version to be pruned in the parallel version before they come up in the queue. The detrimental anomaly occurs when the parallel branch and bound algorithm has to examine more regions than the sequential version. This can occur because the additional processors may expand parts of the tree that were not expanded in the sequential version thereby producing more work.

Having chosen to employ method B, the next major choice is to decide how to manage the list of regions that need to be examined. This is a critical issue because the parallel algorithm will evaluate more regions than the sequential version unless it is always evaluating the most important regions. For the algorithm here, it should always solve the subproblem with the least lower bound. There are two simple ways to achieve this goal. The first is to dedicate one node to act as a manager, leaving the remaining $p - 1$ nodes to act as workers. Each of the worker nodes turns in any regions it generates to the manager and requests a new subproblem to consider. When p is small, giving up one node for management takes away a significant fraction of the machine's computing power, and when p becomes large, the manager may become overwhelmed, causing a sequential or communication bottleneck which in turn causes workers to become idle. The other simple alternative is to have the processors consult with one another between each iteration and decide which problems to solve. However, since the time it takes to solve a subproblem varies, all of the workers are forced to wait until the slowest subproblem is solved.

The alternative to both of these extreme algorithms is to give each node its own separate region list and to adopt a communications strategy to share work as needed and to attempt to make sure all of the processors are solving the most important subproblems. The algorithm presented here is an adaption of the SHL strategy presented by McKeown, Rayward-Smith, and Rush [68]. Each node keeps its own list of regions to analyze, and any time it runs out of work it notifies the other processors by means of an empty list status message. Once it receives some regions from other processors, it sends out a normal status message to prevent processors from sending more regions. Several times per branch and bound loop iteration, a node processes all of its incoming status messages. If after reading all of the status messages, some subset of the nodes are empty; a node will send a fraction of its region list to each idle processor. In order to ensure that all of the nodes are solving the most important work, the nodes are organized into a ring. If 4–6 iterations pass for a node without it receiving any requests from its right neighbor and if it has at least two regions in its list, it will send its second best region to its right neighbor. Thus a node receives unsolicited regions from its left neighbor and sends unsolicited regions to its right neighbor. This causes the high priority work to be dispersed among the nodes. Whenever a node finds a new incumbent, it sends the value to each of the other nodes. This brief overview is explained in greater detail below.

6.1 Detailed Presentation

The algorithm is built around the multiple instructions multiple data (MIMD) paradigm of parallel computing. It is assumed that each processor has its own local memory and that data is shared between processors by message passing. Additionally it is assumed that that the interconnection network and the processors themselves are reliable; this means that messages are not lost or altered, and that processors do not crash or lose contact. Under these assumptions, the algorithm can be implemented on networks of workstations or on dedicated parallel machines like the IBM SP-2, CM-5, or the Intel Hypercube i860. To obtain portability, the algorithm is implemented using the message-passing interface (MPI) [31]. The algorithm always uses asynchronous sends and receives to minimize communications time.

6.1.1 Pre and Post Processing

Before the parallel branch and bound procedure, a number of preprocessing steps need to occur. Node 0 is designated as the manager for this part of the algorithm. It is responsible for processing any command line arguments, parsing the problem definition file, and sending the command line flags and problem definition to each of the nodes. Then all of the processors synchronize before starting the branch and bound procedure. These preprocessing steps are not considered part of the algorithm, so algorithm timing does not start until the branch and bound procedure begins. The optimal distribution of the problem definition and command flags is considered a separate research issue to be addressed by other researchers. The time to distribute this information is generally short compared to the overall run except for particularly easy problems which do not generally need parallel computers to be solved.

After the algorithm terminates, all of the processors synchronize again and the timing stops. The solution and a variety of algorithm statistics are sent to node 0, which is responsible for outputting the solution and statistics. The time to do this is small in comparison to most run times.

6.1.2 Node Data Requirements

Each node maintains the following information throughout the algorithm:

NodeID Numeric node ID (provided by MPI)

NumProcs Number of processors available (provided by MPI)

WorkShare ID of the right neighbor = (NodeID + 1) mod NumProcs

BestObj Incumbent objective function value.

StatusCount The number of status messages sent out (each one is numbered, so receivers can tell which was sent last)

CountsSinceSend The number of communication updates since the last send to node WorkShare

Status[NumProcs] Stores what this node knows about the status of the other nodes

RequestNumber[NumProcs] When the highest number status message from node i indicated an empty list, RequestNumber[i] indicates the number of that status message (each status message from a node is numbered); otherwise, it is 0.

ResponseNumber[NumProcs] When a region is sent in response to an empty list status message from node i, ResponseNumber[i] stores the number of that status message obtained from RequestNumber[i]. RequestNumber[i] is zeroed when a status message with a higher number is received from node i; which is interpreted as an acknowledgment of the regions sent. RequestNumber is used to prevent multiple responses to a particular empty list status message and to prevent a mistaken termination.

HighestRequest[NumProcs] HighestRequest[i] holds the number of the status message received by node i with the highest number. This is included in case messages arrive in a different order than they were sent. Status messages received from node i with numbers less than HighestRequest[i] are ignored.

6.1.3 Message Processing

The branch and bound algorithm itself requires a few changes and the addition of some communications procedures. Both of these modifications will be detailed in this section.

New branch and bound procedures

The first change is to use the incumbent value available from the communication routines rather than the best point found by the particular node. The incumbent value of the communications routine is guaranteed to be less than or equal to the best point found by any particular processor. Each of the procedures presented in the previous chapters require some minor additions. The modified branch and bound loop is as follows with modifications highlighted in *italics*:

Branch and bound loop *(Parallel)*

Initialize search list *Node 0 gets the whole variable domain,*
other nodes start with an empty search list.

While Search list not empty *or the network provides a region:*

0. *Check if other nodes need work.*
1. Remove the top of the search list to be treated as the current region.
2. Choose a point inside the current region using the current point or the midpoint of the region.
3. Analyze the current region (using region analysis procedure).
4. Prune region list if a better feasible point was found.

(Use communication routines for new incumbent value.)

5. If the current region is not pruned due to bound or infeasibility, choose a split variable and a split location and add the two new regions to the region list.

Version 1 of the region analysis procedure is modified as follows:

Region analysis procedure, version 1 *(Parallel)*

1. Call MINOS 5.4 to solve the NLP (11) restricted to $[x^L, x^U]$ using the point provided by the branch and bound loop as a starting place.

1a. Check if other nodes need work.

2. Construct the covering problem at the point found by MINOS 5.4, and return the results to the branch and bound loop.

Version 2 of the region analysis procedure is modified as follows:

Region analysis procedure, version 2 *(Parallel)*

Loop

0. Check if other nodes need work.

1. Construct and solve the covering program at the current point.

1a. Check if other nodes need work.

2. If the covering program is infeasible or if the lower bound allows this region to be pruned, exit the loop.
3. Update the region's lower bound estimate from covering program.

4. If the covering program was stationary ($||\Delta x|| = 0$) then

   If the current point is feasible, exit the loop (optimum found).

   Else, exit because zero search direction.

5. Perform a line search.

6. Check if the current region can be pruned.

Incumbent processing

The parallel algorithm makes use of the best point found by any of the nodes when pruning regions. NewIncumbent messages are sent to exchange this information. Each NewIncumbent message contains a single value which is the new incumbent.

When a new incumbent is found by a node

if (New incumbent < BestObj) then

  BestObj := New incumbent

  Send BestObj to each processor with a message tagged NewIncumbent.

When the incumbent value is needed

while NewIncumbent messages are available:

    Read the new message call its value New incumbent.

    If (New incumbent < BestObj) then

      BestObj = New incumbent

Return BestObj

Region needed from another node

This procedure is called by the branch and bound loop of a node when its region list is empty. If the other nodes are able to provide a region, it returns TRUE and the branch and bound loop continues; otherwise, it returns FALSE and problem have been completed.

Message.type refers to the type of message that was read. A status message contains three pieces of information: the id of the sender (message.id), the node status (message.status), and the status message number (message.number).

When a region is needed

Send an empty list status message to all other node (this
    automatically increments StatusCount by 1).
Status[NodeID] = Empty
CountsSinceSend = 0
Loop
    Return FALSE if Status[i] = Empty and ResponseNumber[i] = 0 for all i
    Wait for a message to arrive.
    While a message is available:
        Read the message.
        switch(Message.type)
        case NewIncumbent:
            Process new incumbent message as above.
        case StatusMessage:
            if (message.number > HighestRequest[message.id]) then

```
            HighestRequest[message.id] = message.number
            if (message.status = Empty) then
              RequestNumber[message.id] = message.number
            else
              RequestNumber[message.id] = 0
            Status[message.id] = message.status
          Update ResponseNumber[message.id].
      case RegionTag:
          Read the region and add it to the list.
          Send a normal status message to all other nodes.
          Status[NodeID] = Normal
          Return TRUE
      End of while
  End of Loop
```

The outer loop is used for checking for termination and the inner loop is used to read all of the incoming messages. These nested loops force all of the available messages to read before termination is checked. This clears all of the messages that may have backed up in the message queue which might contain several empty status message followed by normal status messages. The algorithm does not have a true picture of which processors are idle until it clears all of the incoming messages, so it cannot check for termination after reading each message.

Share work with other nodes

This procedure is very similar to the one just presented, but it is called by a node with at least one region in its list. It reads in any regions that other nodes have sent it and adds them to its list. When indicated by CountsSinceSend, it sends an unsolicited region to its neighbor. It reads all of its incoming status messages, and if it has more than one region in its list, it splits its list among the idle processors.

Work sharing subroutine

```
Process any incoming incumbent message (as above).
while An incoming region message is available:
      Read the message.
      Add the region to the region list.
if (CountsSinceSend >= 12 and regions list has more than two regions) then
      Send the second region to WorkShare.
else
      CountsSinceSend = CountsSinceSend + 1
while An incoming status message is available:
      if (message.number > HighestRequest[message.id]) then
        HighestRequest[message.id] = message.number
        if (message.status = Empty) then
          RequestNumber[message.id] = message.number
        else
          RequestNumber[message.id] = 0
        Status[message.id] = message.status
```

Update ResponseNumber[message.id].

Split the region list among processors with Status[i] = Empty and

ResponseNumber[i] = 0 and itself. Regions are distributed equally, but the node will not send another processor more regions than it keeps for itself. A node with only 1 region on its list does not send any regions.

Again, it is important for the processor to read all of the available status messages to get a true picture of which processors are idle before sending any regions to idle processors.

Comments

These are all of the additional procedures needed. Many of the details of the communications library have been hidden in the presentation, but it communicates the salient features of the algorithm.

Many of the complications in the algorithm are required to treat the problem of status messages arriving in a different order than they were sent and to reliably determine when the algorithm should terminate. A node cannot terminate until it receives a response from each node to which it sent a region in reply to an empty status message.

Chapter 7

Computational Experience

The computational results for the algorithms presented above will be organized into two sections, the first for the sequential algorithm and the second for the parallel algorithm. Both versions have the same core branch and bound algorithm with the parallel version having additional communications and load balancing procedures. The first section presents a detailed analysis of the runtime behavior of the algorithm, and the second section focuses on the speedup and efficiency of the parallelization.

7.1 The Sequential Algorithm

The algorithm presented was implemented in C++ with calls to FORTRAN numerical libraries, and it was tested on four different platforms, a DECstation 3100, a NeXTstation Turbo, a HP 9000/735, and an IBM SP-2. The prototype implementation required approximately 21000 lines of source code including comments and whitespace but not including libraries written by others. This section will present a summary of the results obtained by observing the behavior of the algorithm applied to a variety of problems from the literature.

Tables 1, 2, 3, and 4 give an overview of the problem sizes and characteristics and the runtimes obtained on a HP 9000 Series 700 Model 735 with a clock speed of 99 MHz and 80MB of RAM. Table 5 gives the benchmark results for this machine as reported by HP.

The algorithm was able to solve 47 of the 50 test problems within the arbitrary iteration limit of 25000. In 42 of these cases, it found a result agreeing with the results reported in the problem source, and in 5 cases, it found either better solutions or alternative minima. The value and location of these minima are presented in Appendix B in Tables 23, 24, 25, 26, and 27.

For the three unsolvable problems, the algorithm provides mixed results. For problem fp_3_1, both versions found the solution but were unable to reduce the bound gap to zero within the iteration limit. In the case of problem fp_4_9, neither version of the algorithm found a feasible point. Problem fp_5_1 is not solved within the iteration limit with Version 1, and the runtime was prohibitive for Version 2. Table 6 shows the objective function value found by the algorithm, the bound gap after 25000 iterations, and the number of regions remaining on the list at the end for the unsolved problems.

Figures 8 and 9 show the runtime required to solve a problem versus the number of problem variables and number of bilinearities respectively. Both show a high variance, with the runtime correlating poorly with both. However, the runtime correlates somewhat better with the number of quadratic terms than the number of variables.

Name	n[a]	$x_i x_j$[b]	SVF[c]	NSD[d]	Version 1 Time(s)	Version 1 Iter.	Version 2 Time(s)	Version 2 Iter.	Soln.[e]
fp_2_1	5	5	0	0	1.48	33	1.23	37	$\surd$
fp_2_2	6	5	0	0	0.03	1	0.02	1	$\surd$
fp_2_3	13	4	0	1	1.82	16	0.11	3	$\surd$
fp_2_4	6	1	0	1	0.11	3	0.08	7	$\surd$
fp_2_5	10	7	0	0	1.24	31	0.81	21	$*$
fp_2_6	10	10	0	0	3.91	31	2.73	27	$\surd$
fp_2_7_1	20	20	0	0	275.68	643	363.55	663	$\surd$
fp_2_7_2	20	20	0	0	265.86	615	275.10	515	$\surd$
fp_2_7_3	20	20	0	0	751.67	971	642.74	835	$\surd$
fp_2_7_4	20	20	0	0	198.26	623	213.15	473	$\surd$
fp_2_7_5	20	20	0	0	1168.83	1909	745.30	1611	$*$
fp_2_8	24	24	0	1	48.18	53	148.49	141	$*$
fp_3_1	8	5	0	2	1278.21	25000	4837.56	25000	$\times$
fp_3_2	5	8	0	2	1.26	27	21.15	40	$\surd$
fp_3_3	6	7	0	0	1.01	23	0.40	7	$\surd$
fp_3_4	3	9	0	0	245.90	9213	595.69	6269	$*$
fp_4_3	4	2	2	0	0.23	15	0.12	5	$\surd$
fp_4_4	4	2	2	0	0.34	25	0.30	19	$\surd$
fp_4_5	6	3	3	0	0.62	35	0.80	29	$*$
fp_4_6	2	1	2	0	0.15	13	0.32	11	$\surd$
fp_4_7	2	2	1	1	0.05	3	0.37	17	$\surd$
fp_4_9	11[f]	16	8	0	9357.13	25000	19472.80	25000	$\times$
fp_5_1	48	44	0	9	192930.82	25000	n/a	n/a	$\times$

[a]The number of variables

[b]The number of bilinearities

[c]The number of single variable nonlinear functions. Each of these is counted as a quadratic variable too.

[d]The number of null space dimensions at the solution.

[e]$\surd$ means the solution agrees with published values, $*$ means the algorithm found a better or different solution, and $\times$ means the algorithm did not find a solution within the iteration limit.

[f]Application of the algorithm to this problem required 11 additional variables.

Table 1: Problem Set 1 from Floudas and Pardalos

Name	n[a]	x_ix_j[b]	SVF[c]	NSD[d]	Version 1 Time(s)	Version 1 Iter.	Version 2 Time(s)	Version 2 Iter.	Soln.[e]
f_a	2	1	0	1	0.46	35	0.92	23	√
f_b	2	1	0	0	0.03	1	0.01	1	√
f_c	4	4	0	0	0.04	1	0.03	1	√
f_d	20	20	0	0	49.93	87	30.67	103	√
f_e	6	6	0	0	1.38	23	0.17	3	√
f_f	2	2	0	1	0.21	9	0.27	9	√
f_g_I	9	2	0	1	0.08	3	0.52	6	√
f_g_II	9	2	0	1	0.31	3	0.29	3	√
f_g_III	9	2	0	0	0.35	7	1.91	13	√
f_wingo	1	1	3	1	0.32	27	0.24	27	√

[a]The number of variables

[b]The number of bilinearities

[c]The number of single variable nonlinear functions. Each of these is counted as a quadratic variable too.

[d]The number of null space dimensions at the solution.

[e]√ means the solution agrees with published values, ∗ means the algorithm found a better or different solution, and × means the algorithm did not find a solution within the iteration limit.

Table 2: Problem Set 2 from Floudas and Visweswaran

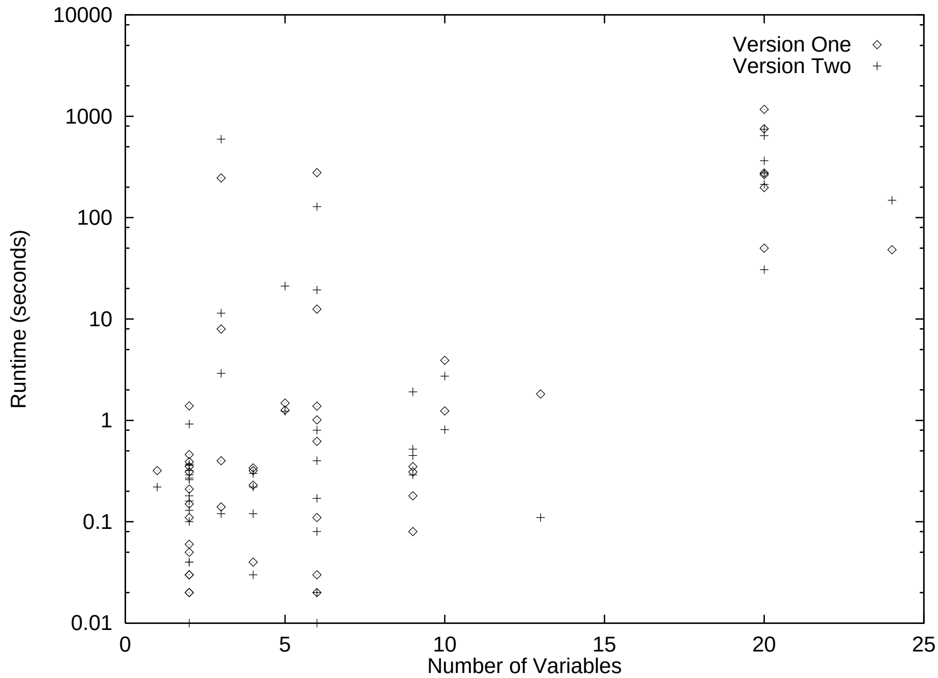


Figure 8: Runtime versus number of variables

Name	n^a	${x_i x_j}^b$	SVF[c]	NSD[d]	Version 1 Time(s)	Version 1 Iter.	Version 2 Time(s)	Version 2 Iter.	Soln.[e]
s_1	2	3	0	0	0.35	19	0.36	19	$\surd$
s_1b	2	3	0	0	0.39	19	0.29	21	$\surd$
s_1c	2	3	0	0	0.11	7	0.10	7	$\surd$
s_1d	2	3	0	0	0.31	17	0.26	13	$\surd$
s_2	2	1	0	1	0.02	1	0.16	5	$\surd$
s_2b	2	1	0	1	1.39	119	0.13	7	$\surd$
s_2c	2	1	0	1	0.02	1	0.04	1	$\surd$
s_2d	2	1	0	1	0.03	1	0.04	1	$\surd$
s_3	2	2	1	1	0.06	3	0.18	3	$\surd$
s_4	3	3	2	0	0.40	23	2.91	56	$\surd$
s_5	4	2	2	0	0.32	19	0.22	17	$\surd$
s_6	9	2	0	1	0.18	3	0.45	3	$\surd$

[a]The number of variables

[b]The number of bilinearities

[c]The number of single variable nonlinear functions. Each of these is counted as a quadratic variable too.

[d]The number of null space dimensions at the solution.

[e]$\surd$ means the solution agrees with published values, $*$ means the algorithm found a better or different solution, and $\times$ means the algorithm did not find a solution within the iteration limit.

Table 3: Problem Set 3 from Swaney

Name	n^a	${x_i x_j}^b$	SVF[c]	NSD[d]	Version 1 Time(s)	Version 1 Iter.	Version 2 Time(s)	Version 2 Iter.	Soln.[e]
diet	6	0	0	0	0.02	1	0.01	1	$\surd$
e_1	3	3	0	2	0.14	7	0.12	7	$\surd$
w	6	8	5	0	278.13	4129	128.25	885	$\surd$
w_b	6	5	8	0	12.51	323	19.34	128	$\surd$
w_c	3	5	8	0	7.97	307	11.43	129	$\surd$

[a]The number of variables

[b]The number of bilinearities

[c]The number of single variable nonlinear functions. Each of these is counted as a quadratic variable too.

[d]The number of null space dimensions at the solution.

[e]$\surd$ means the solution agrees with published values, $*$ means the algorithm found a better or different solution, and $\times$ means the algorithm did not find a solution within the iteration limit.

Table 4: Problem Set 4: Miscellaneous Problems

SPECint92	109
SPECfp92	168
MIPS	125
MFLOPS (dp)	45

Table 5: Benchmarks for HP 9000 Series 700 Model 735 at 99 MHz

	Version 1			Version 2		
Name	Objective	Gap	Regions	Objective	Gap	Regions
fp_3_1	7049.25	108.98	2443	7049.25	135.59	2181
fp_4_9	None	∞	3693	None	∞	2189
fp_5_1	1.567	0.132931	5868	n/a	n/a	n/a

Table 6: Results for unsolved problems

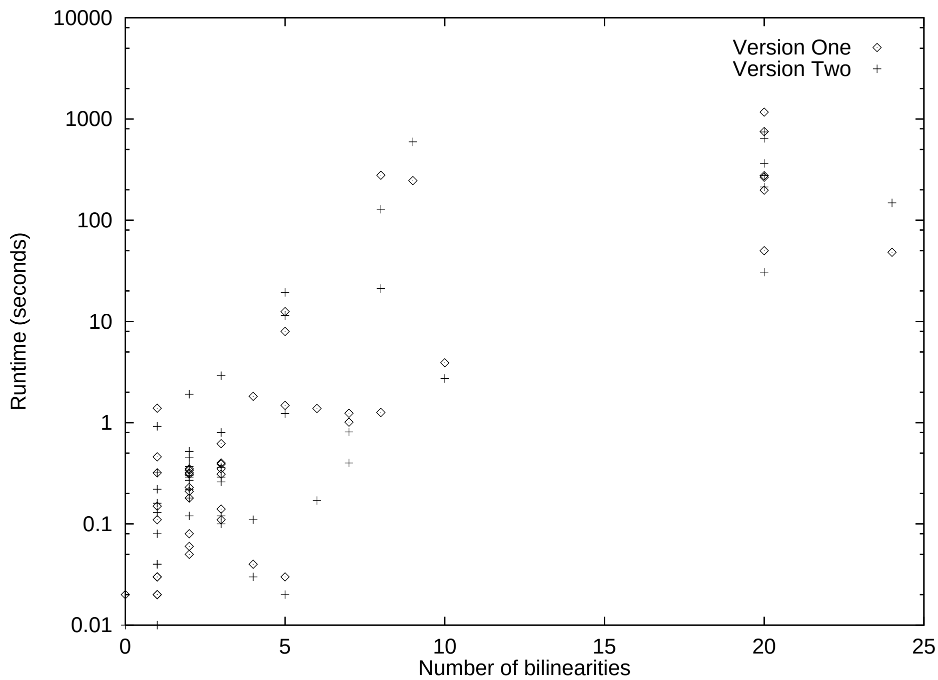


Figure 9: Runtime versus number of bilinearities

Version 1 and version 2 of the algorithm have comparable runtimes, and neither is clearly better than the other. Version 1 required less time than version 2 in 22 of the 47 solvable problems. The combined time required to solve all of the problems for version 1 is 3322.09 seconds, and the combined time for version 2 is 3211.76 seconds. Version 2 requires fewer iterations in 21 of the 47 solvable problems, and they tie on 14 problems. Version 1 requires a total of 19478 iterations to solve all of the problems, and version 2 requires 12223 iterations.

It is also useful to compare function, constraint and constraint gradient evaluations for the two algorithms. Tables 7, 8, 9, and 10 show this comparison. Version 2 of the algorithm requires fewer objective function evaluations in 32 of the 47 problems, fewer constraint evaluations in 40 of the 47 problems, and fewer constraint gradient evaluations in 44 of the 47 problems. Table 11 compares the sum of the evaluations for all of the problems for each version which highlights the observation that version 2 requires almost an order of magnitude fewer constraint evaluations and more than an order of magnitude fewer constraint gradient evaluations. Version 1 is solving more quick subproblems, and version 2 is solving fewer slow subproblems.

The results of the runs of both algorithms have been grouped into characteristic behaviors, and some examples of each characteristic behavior will be presented. The runs have been grouped into the following categories: trivial runs, runs where the log of the bound gap decreases nearly linearly, runs where the log of the bound gap decreases superlinearly, and poor runs. All of the graphs are in terms of the scaled objective function. Some problems appear in different groups for the different algorithm versions.

Name	Version 1			Version 2		
	Obj.[a]	Const.[b]	Grad.[c]	Obj.	Const.	Grad.
fp_2_1	4162	4195	4224	172	181	54
fp_2_2	10	11	12	2	2	2
fp_2_3	218	234	243	22	23	4
fp_2_4	137	140	142	35	38	11
fp_2_5	229	273	290	64	65	20
fp_2_6	4463	4814	4836	74	74	24
fp_2_7_1	7647	40855	41183	3614	3760	679
fp_2_7_2	13347	38686	39014	3565	3775	697
fp_2_7_3	43323	154888	155372	4740	4886	770
fp_2_7_4	13425	35490	35818	2899	3028	490
fp_2_7_5	78500	245840	246859	5653	5735	1390
fp_2_8	4651	4709	4736	917	994	197
fp_3_2	67	1327	1339	624	673	118
fp_3_3	833	1989	2001	41	45	11
fp_3_4	87141	334185	341338	57562	64512	11849
fp_4_3	49	69	77	39	39	9
fp_4_4	145	175	192	93	98	22
fp_4_5	247	346	366	240	260	40
fp_4_6	121	213	220	241	261	46
fp_4_7	6	91	92	94	100	17

[a] Objective function evaluations
[b] Constraint evaluations
[c] Constraint gradient evaluations

Table 7: Function evaluation comparison for problem set 1

	Version 1			Version 2		
Name	Obj.[a]	Const.[b]	Grad.[c]	Obj.	Const.	Grad.
f_a	176	398	416	352	423	89
f_b	43	44	45	2	2	2
f_c	13	16	17	2	2	2
f_d	4454	8882	8925	331	336	70
f_e	933	3342	3354	17	19	6
f_f	252	440	446	106	122	24
f_g_I	24	84	86	39	42	10
f_g_II	19	155	157	13	15	9
f_g_III	46	499	503	69	72	19
f_wingo	202	406	420	129	134	20

[a]Objective function evaluations
[b]Constraint evaluations
[c]Constraint gradient evaluations

Table 8: Function evaluation comparison for problem set 2

	Version 1			Version 2		
Name	Obj.[a]	Const.[b]	Grad.[c]	Obj.	Const.	Grad.
s_1	350	541	550	123	130	25
s_1b	520	768	778	97	99	19
s_1c	21	48	51	26	27	8
s_1d	279	422	430	62	65	13
s_2	3	3	4	75	85	15
s_2b	633	1038	1124	29	32	9
s_2c	3	3	4	6	8	4
s_2d	3	3	4	8	11	5
s_3	5	100	101	24	31	9
s_4	76	389	401	524	577	95
s_5	97	125	134	62	64	12
s_6	12	84	86	16	18	6

[a]Objective function evaluations
[b]Constraint evaluations
[c]Constraint gradient evaluations

Table 9: Function evaluation comparison for problem set 3

Name	Version 1 Obj.[a]	Const.[b]	Grad.[c]	Version 2 Obj.	Const.	Grad.
diet	10	10	11	2	2	2
e_1	134	198	202	39	41	12
w	4683	61969	64406	9632	11105	2441
w_b	1223	6991	7160	1279	1462	301
w_c	1734	4619	4778	1439	1537	273

[a]Objective function evaluations
[b]Constraint evaluations
[c]Constraint gradient evaluations

Table 10: Function evaluation comparison for problem set 4

	Version 1	Version 2
Function evaluations	274669	95194
Constraint evaluations	960107	105010
Constraint gradient evaluations	972947	19950

Table 11: Cumulative totals for function evaluations

The first category consists of runs which took less than fifteen iterations to solve for a particular algorithm. Figures 10 and 11 show the bounds as a function of iteration number for problem f_g_III for versions 1 and 2 of the algorithm respectively. In both cases, the bounds converge in a few iterations.

The second category are those where the log of the bound gap decreases approximately linearly with the iteration number. Applying version 1 of the algorithm to problem fp_2_7_5 demonstrates this behavior on a problem with 20 variables and 20 quadratic terms, and applying version 2 of the algorithm to problem s_1 demonstrates this performance on a problem with 2 variables and 1 quadratic term. Figures 12, 13, and 14 show the bounds, bound gap, and region size versus iteration number for fp_2_7_5 using version 1, and Figures 15, 16, and 17 show the analogous results for s_1.

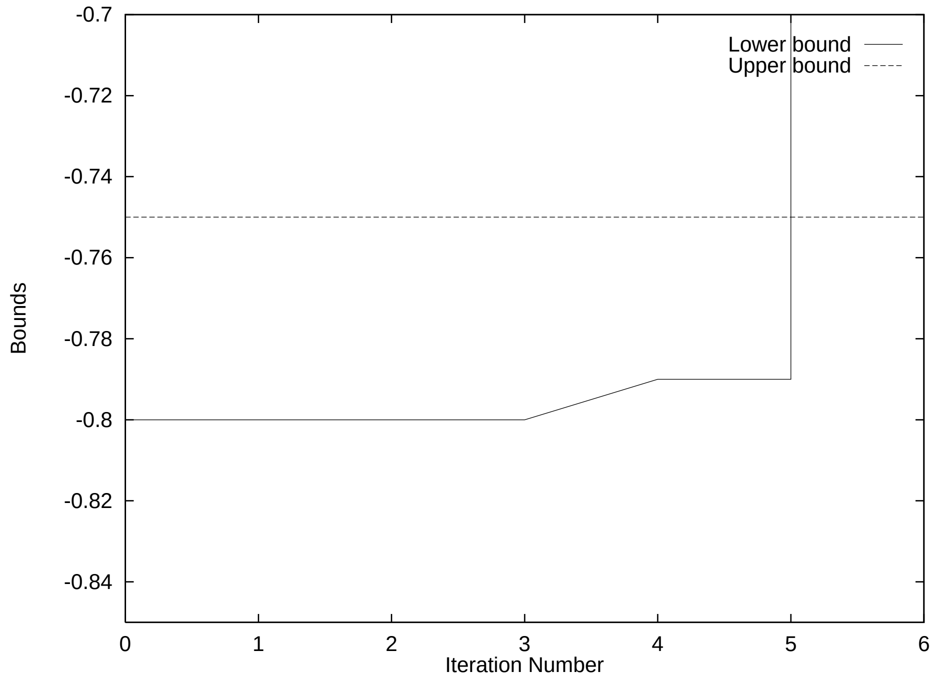


Figure 10: Bounds versus iteration number for typical rapid convergence problem (f_g_III, version 1)

The next category are those where the log of the bound gap decreases superlinearly with respect to the iteration number. Applying version 2 of the algorithm to problem fp_2_7_3 gives a typical example of this as shown in Figures 18, 19 and 20. The bound gap decreases superlinearly on a semi-log plot.

Problem fp_3_4 displays another kind of superlinear decrease, and the behavior is virtually the same for both algorithms. Because the behaviors are similar, only version 1 of the algorithm is shown. Figures 21, 22, and 23 show the bounds, bound gap, and region size versus iteration number. The lower bound increases slowly and linearly with respect to iteration number, which causes the semi-log plot to have negative curvature. This was the only problem to display this kind of behavior. The difficulty is not caused by problem size because the problem only

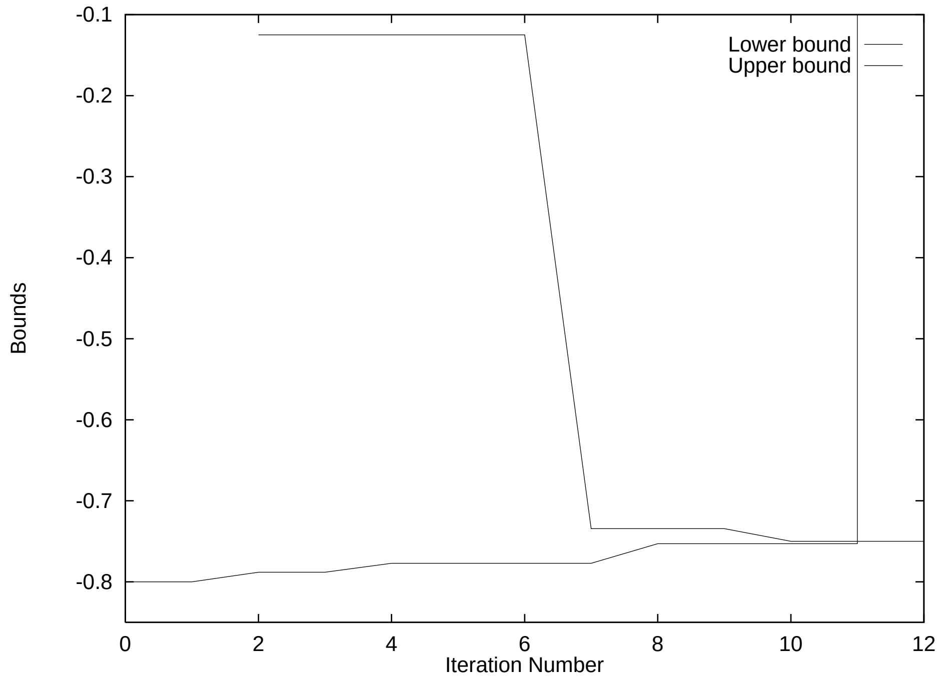


Figure 11: Bounds versus iteration number for typical rapid convergence problem (f_g_III, version 2)

has three variables and nine quadratic terms, and it has a linear objective functions, two linear constraints, and one reverse convex constraint.

Another interesting difference in this problem is the region size versus iteration number graph, Figure 23. Most problems show a much greater reduction in the region size over that many iterations, which suggests that the difficulty may lie in the splitting method. Most of the problems are able to use the split recommended by the sensitivity analysis, referred to here as the "best split," most of the time; the alternative is referred to here as the "widest split." Table 12 shows the percentages of best and widest splits for problems using more than 5% widest split for one of the versions. The runtimes in Tables 1, 2, 3, and 4, tend to be higher for problems with a high percentage of widest splits. This suggests that when the algorithm has

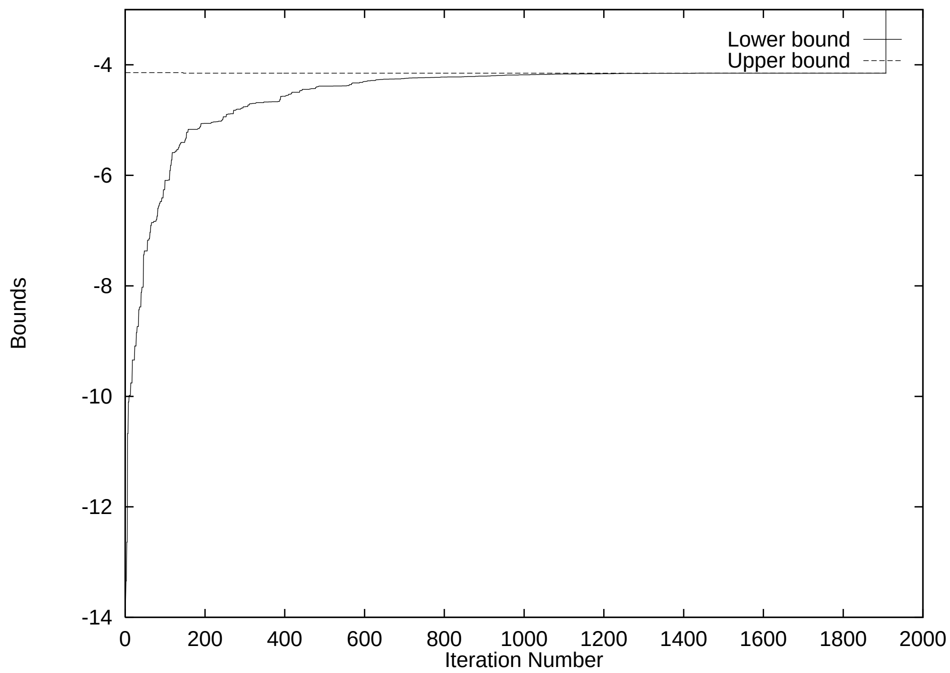


Figure 12: Bounds versus iteration number for typical linear convergence problem (fp_2_7_5, version 1)

to resort to a widest split for certain variables, the efficiency suffers.

One of the adjustable parameters in the algorithm is the ratio of variable widths required to override the best split. The normal value is 0.01, and the best split is overridden while the following inequality is satisfied:

$$\frac{x^U_{\text{best}} - x^L_{\text{best}}}{\bar{x}_{\text{best}} - \underline{x}_{\text{best}}} \leq (\text{Override}) \cdot \frac{x^U_{\text{widest}} - x^L_{\text{widest}}}{\bar{x}_{\text{widest}} - \underline{x}_{\text{widest}}}$$

Table 13 shows how varying the widest override parameter affects the runtime and the number of regions searched for the problems in Table 12. In the first two cases of problem fp_3_1, the algorithm does not find a solution. For Override = 0.001, problem fp_3_1 terminates with a bound gap of 0.158927, and for Override = 0.01, it terminates with a bound gap of 0.135591. For the third case, Override = 0.1, the algorithm solves fp_3_1. For Override = 0.001, problem fp_3_4 exceeds the iteration

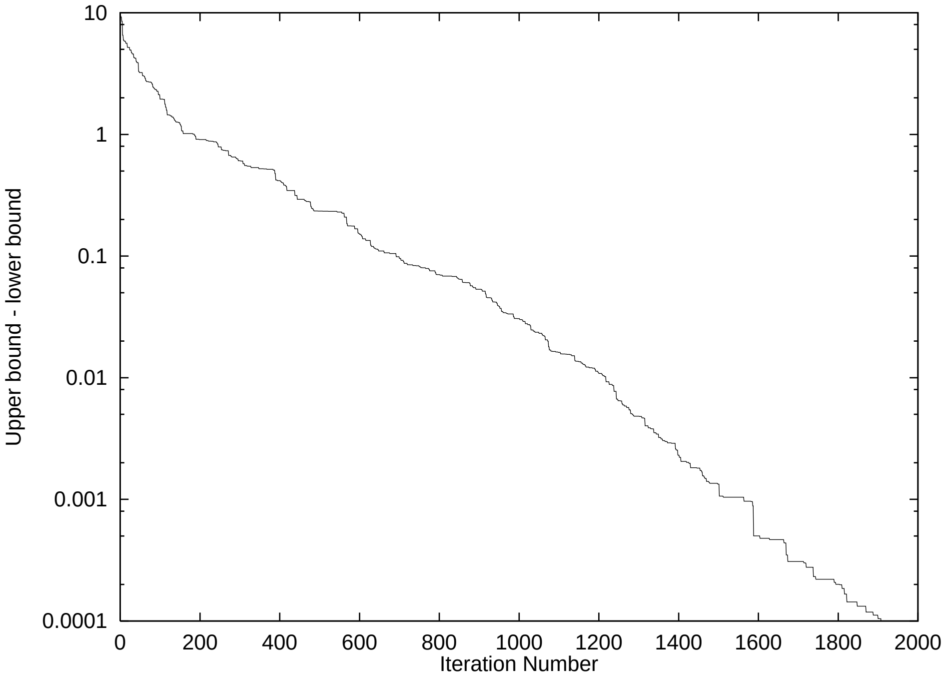


Figure 13: Bound gap versus iteration number for typical linear convergence problem (fp_2_7_5, version 1)

	Version 1		Version 2	
Name	Widest	Best	Widest	Best
fp_3_1	0.3558	0.6442	0.3500	0.6500
fp_3_4	0.4870	0.5130	0.4650	0.5350
fp_4_9	0.5781	0.4219	0.3965	0.6035
fp_5_1	0.1438	0.8562	n/a	n/a
s_2b	0.1017	0.8983	0.0000	1.0000
s_4	0.0909	0.9091	0.4722	0.5278
w	0.5804	0.4196	0.6240	0.3760
w_b	0.3540	0.6460	0.4318	0.5682
w_c	0.4314	0.5686	0.4302	0.5698

Table 12: Problems which use the widest rule over 5% of the time

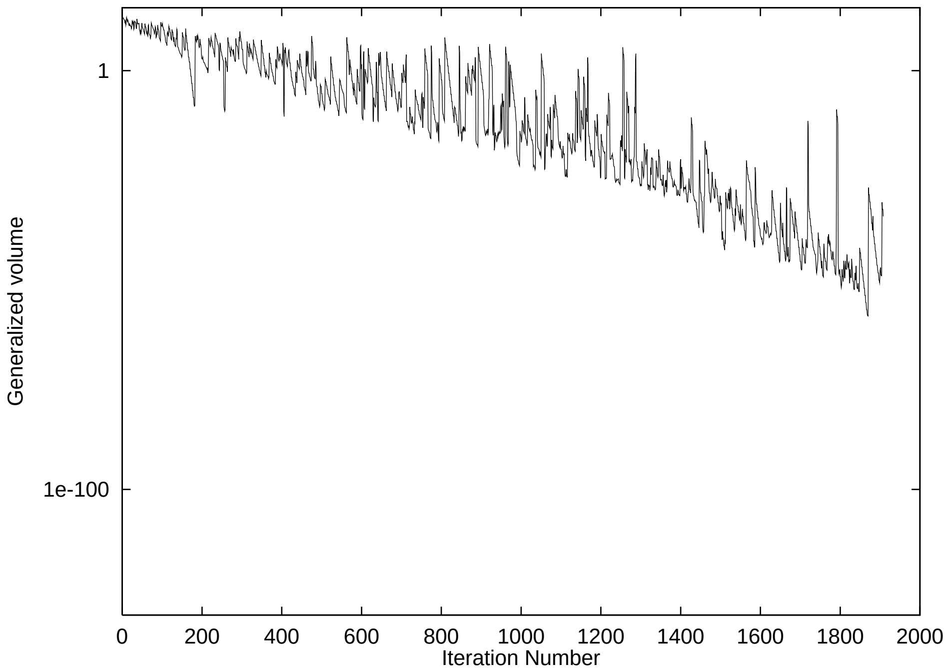


Figure 14: $\prod_{i=1}^{n}(x_i^U - x_i^L)$ versus iteration number for typical linear convergence problem (fp_2_7_5, version 1)

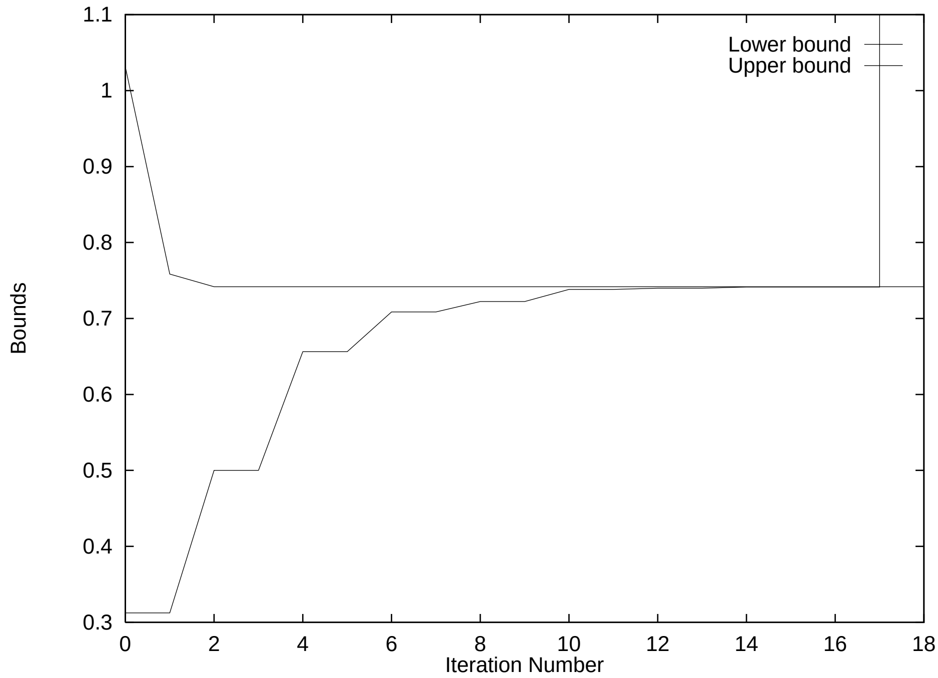


Figure 15: Bounds versus iteration number for typical linear convergence problem (s_1, version 2)

limit before the solution is verified; however, as the override factor increases the problem solves faster. Problem fp_4_9 does not find a feasible point in all three cases, so its bound gap is infinite. Most of these problems benefit from having a higher override factor, which branches using the widest rule more frequently, thereby showing that the sensitivity based selection rule is not providing good splits for these problems.

Table 14 shows the same information for the remaining problems when the change in override factor caused more than a negligible change in the performance. Most of the problems are unaffected by the change in override factor, and among those that are, there is no consistent trend.

In the case of problem fp_3_4, the slow convergence is due to the splitting

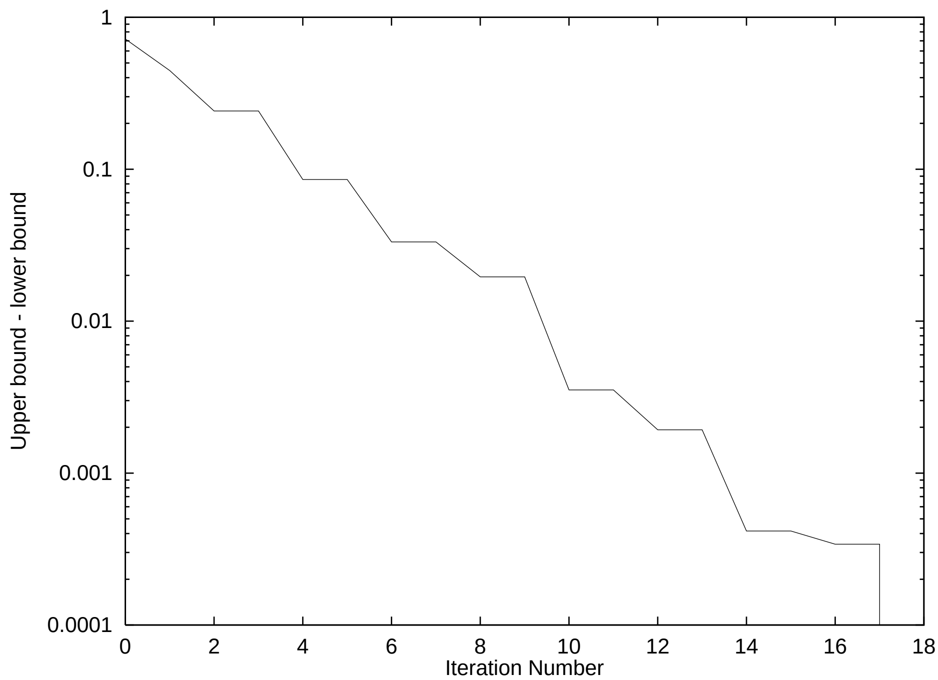


Figure 16: Bound gap versus iteration number for typical linear convergence problem (s_1, version 2)

	Override = 0.001		Override = 0.01		Override = 0.1	
Name	Time (s)	Regions	Time (s)	Regions	Time (s)	Regions
fp_3_1	4545	25000	4837.56	25000	2568.19	8563
fp_3_4	1454.04	25000	595.69	6269	16.96	187
fp_4_9	21316.41	25000	19472.80	25000	25309.63	25000
s_2b	0.13	7	0.13	7	0.13	7
s_4	3.06	60	2.91	56	2.92	52
w	2943.41	21407	128.25	885	46.61	271
w_b	17.48	111	19.34	128	13.35	91
w_c	12.15	150	11.43	129	10.60	96

Table 13: Effect of the override parameter on runtime for version 2

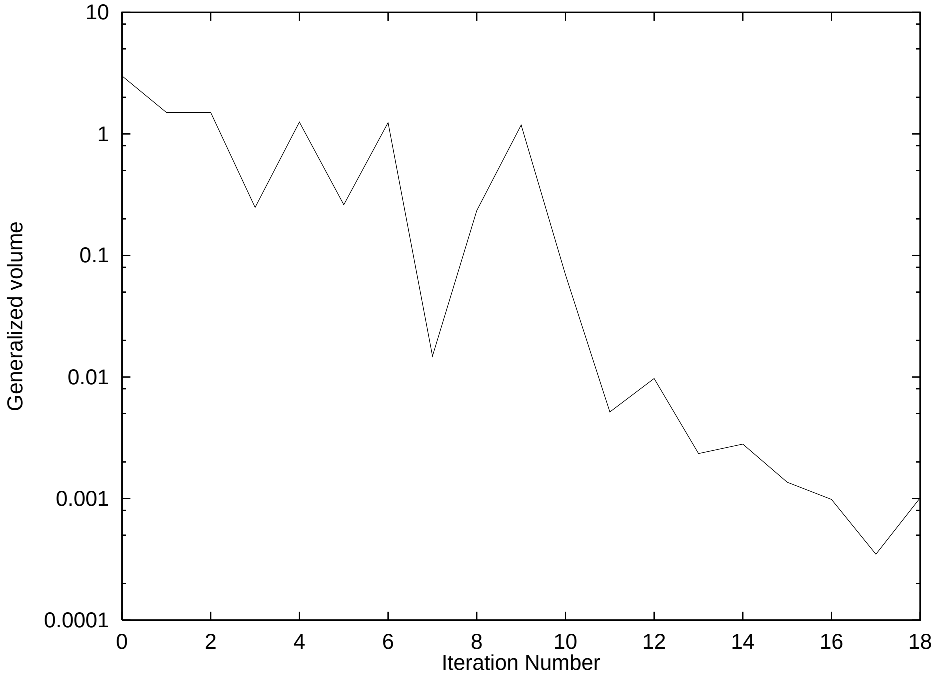


Figure 17: $\prod_{i=1}^{n}(x_i^U - x_i^L)$ versus iteration number for typical linear convergence problem (s_1, version 2)

Name	Override = 0.001 Time (s)	Override = 0.001 Regions	Override = 0.01 Time (s)	Override = 0.01 Regions	Override = 0.1 Time (s)	Override = 0.1 Regions
fp_2_7_1	361.03	663	363.55	663	362.35	713
fp_2_7_2	272.63	515	275.10	515	273.16	515
fp_2_7_3	617.85	835	642.74	835	610.08	753
fp_2_7_4	212.57	473	213.15	473	215.96	523
fp_2_7_5	747.42	1623	745.30	1611	790.77	1771
fp_2_8	145.50	141	148.49	141	163.71	227
fp_3_2	21.14	40	21.15	40	13.10	35
f_d	30.58	103	30.67	103	35.93	121

Table 14: Effect of the override parameter on runtime for version 2

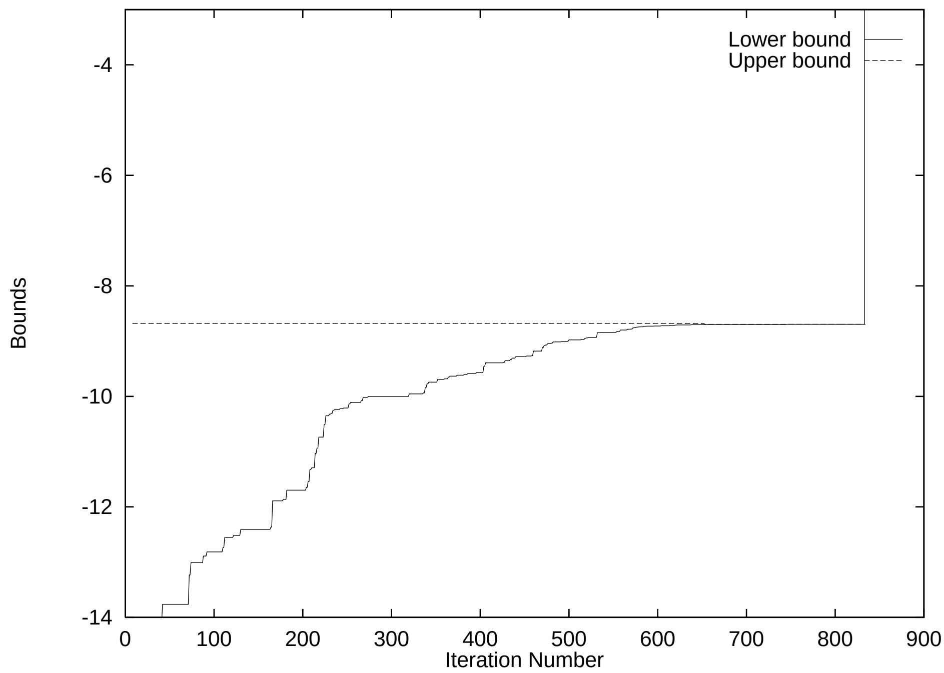


Figure 18: Bounds versus iteration number for typical superlinear convergence problem (fp_2_7_3, version 2)

rule. The best splitting rule almost always recommends variable one or two; however, the algorithm requires variable three to be split to verify the solution. Generally, variable three is only split when the best split is overridden (i.e. when variable three's region width is 100 times that of the recommended variable). The sensitivity analysis usually misses the dependence on variable three bounds because the McCormick constraints (161–164) are active. For example, if constraint (161) is active for bilinearity $2, 3$ with a Lagrange multiplier of $u_{2,3}$, the effect on the objective of a change in x_3^U in this constraint is estimated by

$$\Delta\text{Objective} \approx u_{2,3}(\Delta x_2 - \Delta x_2^U)(\Delta x_3^{U\prime} - \Delta x_3^U)$$

where $\Delta x_3^{U\prime}$ is the new bound for Δx_3 and Δx_3^U is the old bound. When $\Delta x_2 = \Delta x_2^U$, which occurs frequently in problem fp_3_4, the dependence on

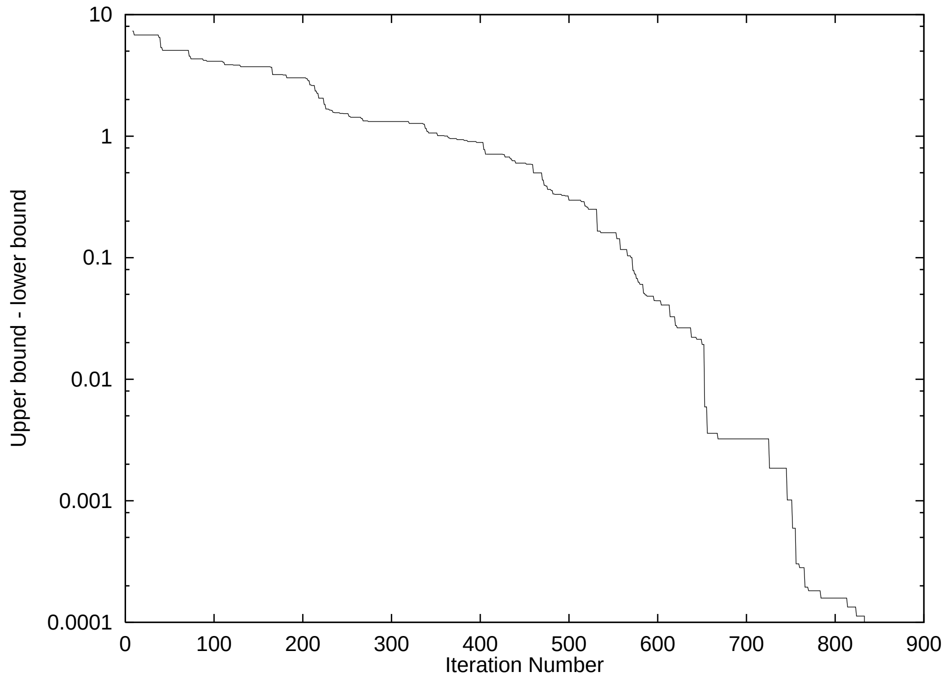


Figure 19: Bound gap versus iteration number for typical superlinear convergence problem (fp_2_7_3, version 2)

x_3^U disappears. The sensitivity analysis predicts changes assuming that the LP basis remains the same, so it cannot predict how a change in the bounds of x_3 will affect the LP basis and consequently the objective function. This kind of problem keeps the sensitivity based selection rule from choosing variable three. If $\check{x}$, the point around which the covering program is constructed, is in the interior of the region (ie. not at a variable bound), McCormick's bounds, constraints (161–164), will always have a bound gap as shown in Figure 3, so ultimately, those constraints must leave the basis and be replaced by the constraint space or null space quadratic constraints. It may be possible to improve the selection rule by using a more sophisticated and computationally intensive sensitivity analysis which can account for potential changes in the LP basis [25].

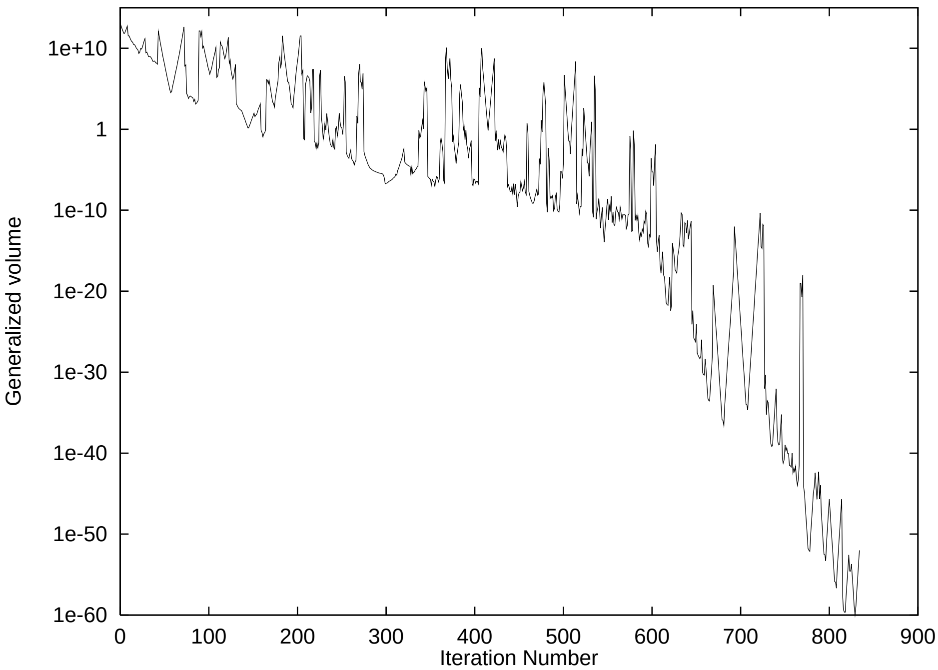


Figure 20: $\prod_{i=1}^{n}(x_i^U - x_i^L)$ versus iteration number for typical linear convergence problem (fp_2_7_3, version 2)

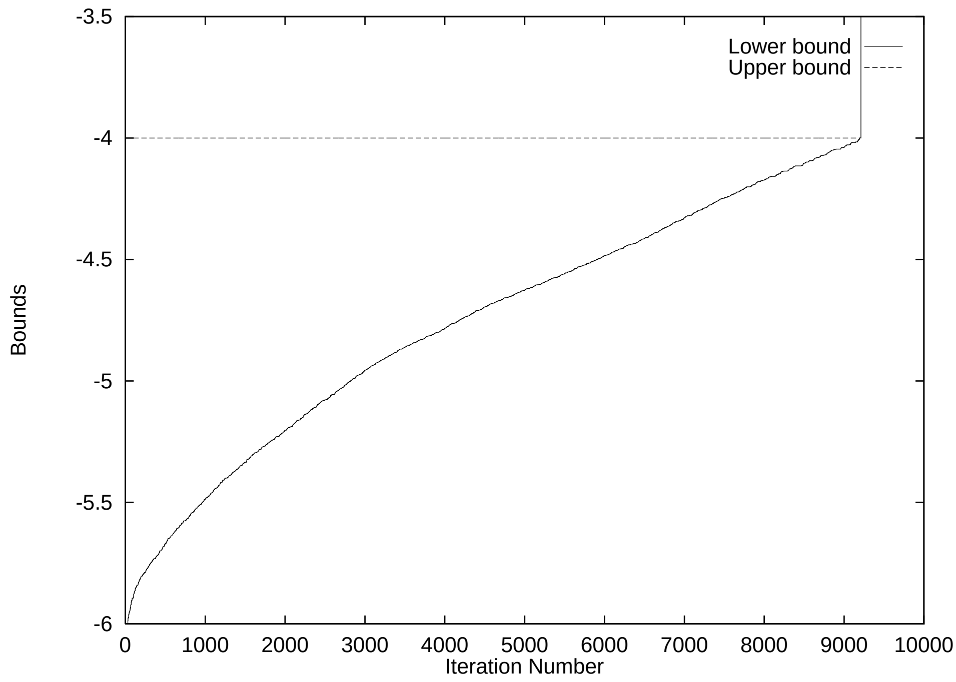


Figure 21: Bounds versus iteration number for problem fp_3_4, version 1

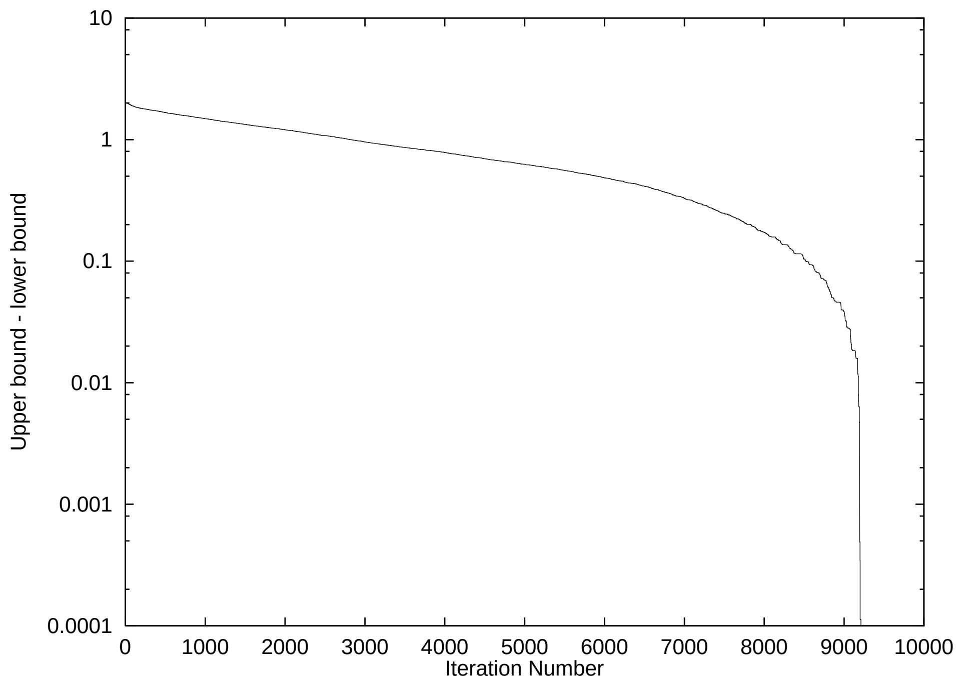


Figure 22: Bound gap versus iteration number for problem fp_3_4, version 1

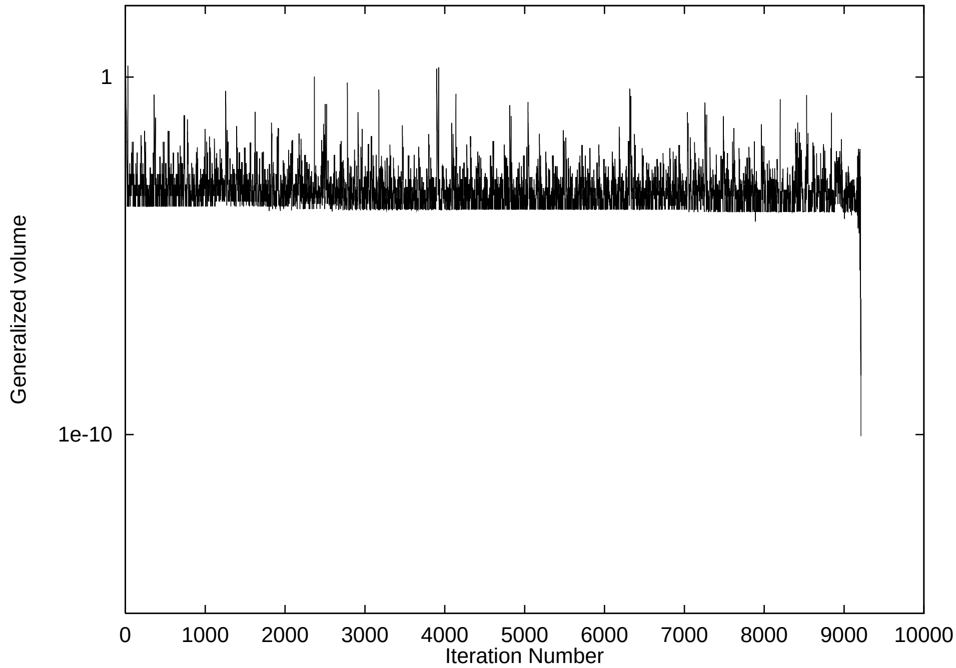


Figure 23: $\prod_{i=1}^{n}(x_i^U - x_i^L)$ versus iteration number for problem fp_3_4, version 1

Figures 24 and 25 illustrates how the algorithm can proceed for a number of iterations without any improvement in the lower bound. The flat spaces in the bound graph can be explained by a feature of the splitting strategy. The algorithm always examines the region with the least lower bound. When it splits, the splitting strategy may select a variable that produces one region with an increased lower bound and another with no increase. This happens when the solution of the covering program depends strongly on either the upper or lower bound but not both. Because of this feature, the splitting strategy may have to split several times before it picks a variable that will raise the least lower bound. The criteria for choosing a variable, (221), favors variables where both bounds are significant, but it may still choose variables which do not improve the least lower bound.

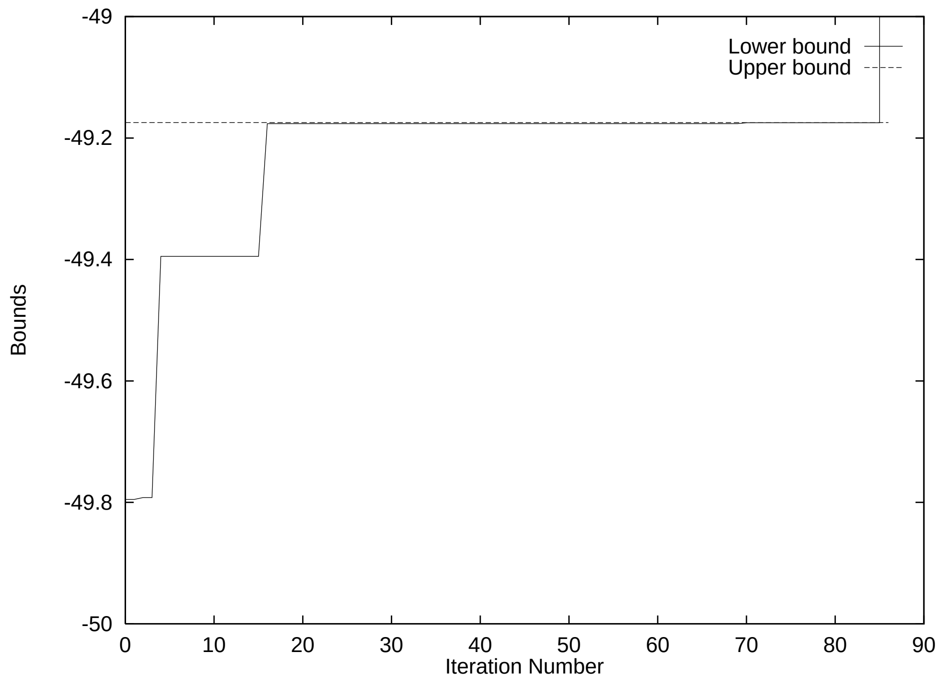


Figure 24: Bounds versus iteration number for problem f_d, version 1

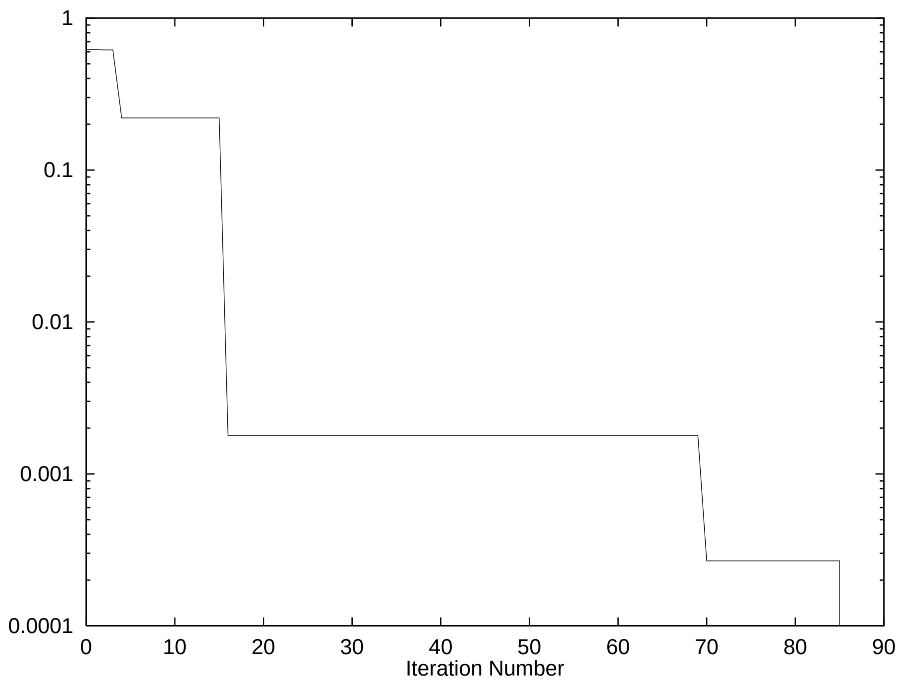


Figure 25: Bound gap versus iteration number for problem f_d, version 1

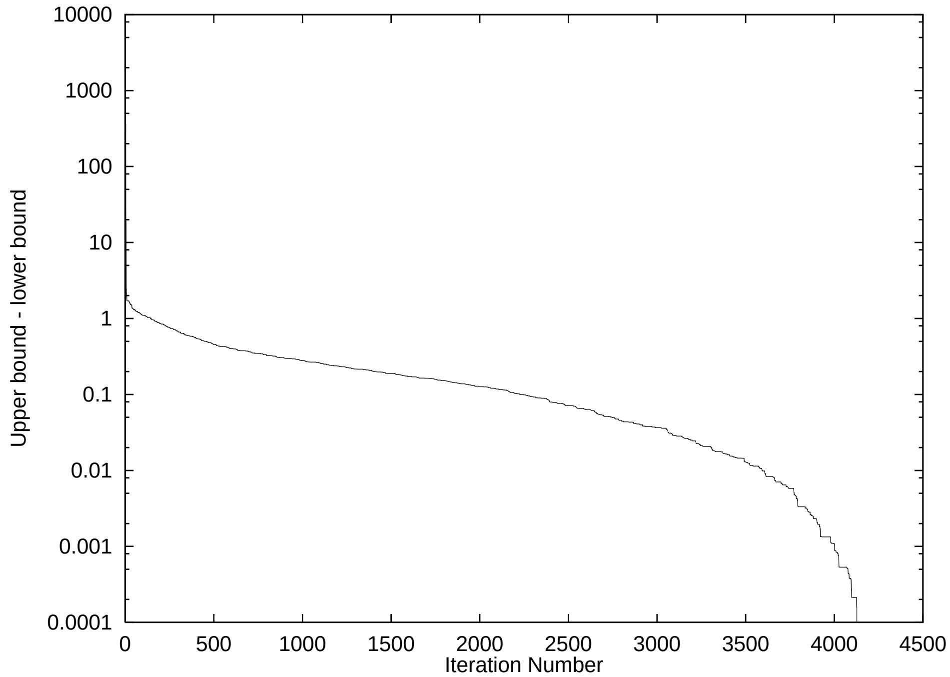


Figure 26: Bound gap versus iteration number for problem w, version 1

The last category of problems are ones where an algorithm performed poorly. Applying version 1 to problems w and s_2b results in slow convergence. In the case of problem w shown by Figure 26, the difficulty seems to be due to the unbounded derivative at $x^{0.6}$ at $x = 0$. At low values of x, $x^{0.6}$ is replaced with a polynomial, but it still has a first and second derivative which causes the underestimating quadratic to fit poorly. The poor underestimation causes it to require a very small region to verify the minimum.

Version 1 requires 119 iterations to solve problem s_2b, which is much larger than the 7 iterations required by version 2, and the performance of version 1 is shown in Figure 27. Of the 119 regions that the algorithm examines only 4 of them contain the global minimum. The remaining 115 are needed to eliminate the area that does not contain the solution, which is more than it ought to take

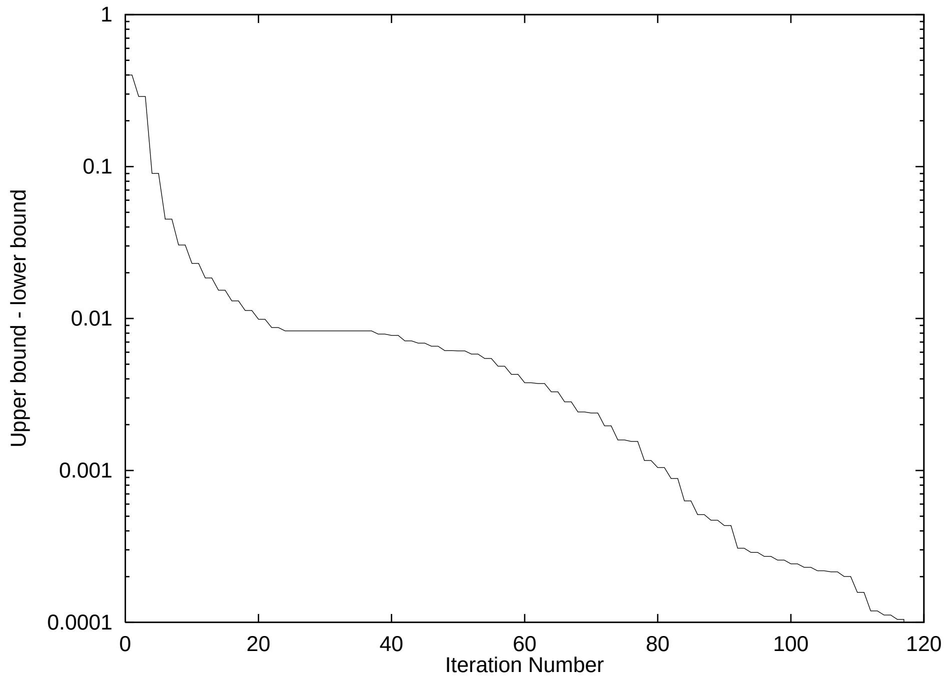


Figure 27: Bound gap versus iteration number for problem s_2b, version 1

considering the size of the problem and the overall size of the variable domain. The excess is due to a split very near the global optimum and looseness in the full rank program (187). The split near the global optimum causes the regions to have lower bounds very close to the global optimum, and the looseness in the full rank program is sufficient to keep them from getting pruned. The looseness in the full rank program comes because of difficulty in choosing constraints (197). There are three variables and only two constraints, so one of the variables does not have a constraint of the form of (197) to help enforce complementarity of its positive and negative component. This problem might be solved by generating more than one constraint of the form (197) for a particular original constraint.

It is also interesting to look at an example of when the algorithm failed. Versions 1 and 2 of the algorithm have very similar performance for problem fp_3_1. Figures

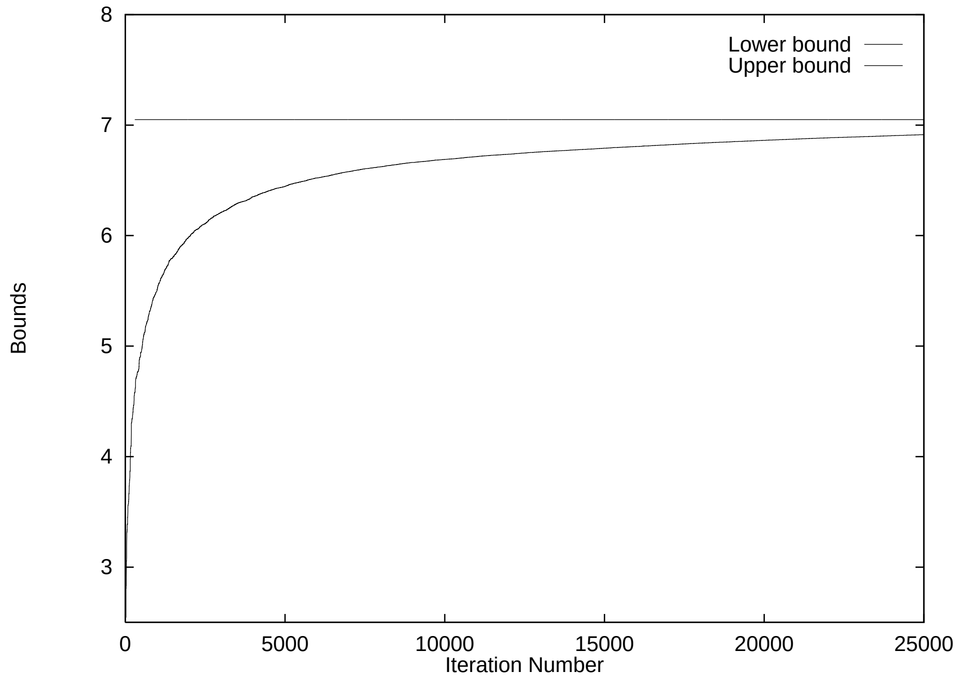


Figure 28: Bounds versus iteration number for problem fp_3_1, version 2

28–30 show the performance of version 2 of the algorithm. It appears that the lower bound is asymptotically approaching the upper bound at a slow rate.

The splitting rule is not recommending the best sequence of splits to improve the lower bound. The algorithm logs show that this problem is suffering from the same difficulty as fp_3_4. The sensitivity with respect to some variables is disappearing because the McCormick envelopes indicate no dependence on some variable bounds. The sensitivity based splitting rule is usually recommending three of the eight variables, and the remaining five are usually split by the widest split override. Table 15 shows the frequency of different types of splits for each variable. It is also interesting to note the small percentage of the best splits which are predicted to eliminate a region.

Another difficulty with problem fp_3_1 is that the branch and bound list size

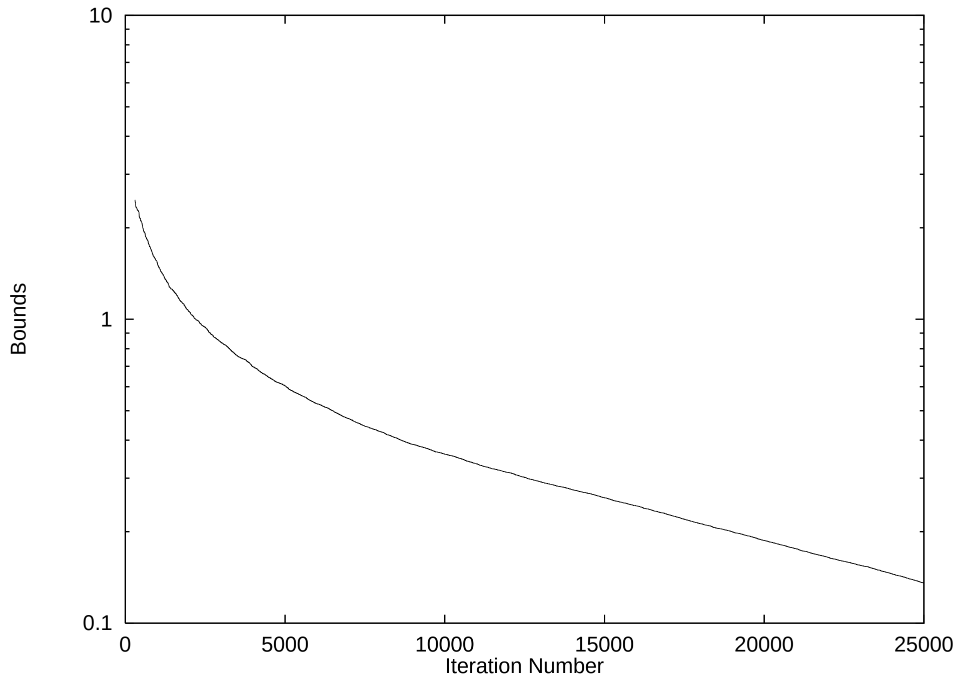


Figure 29: Bound gap versus iteration number for problem fp_3_1, version 2

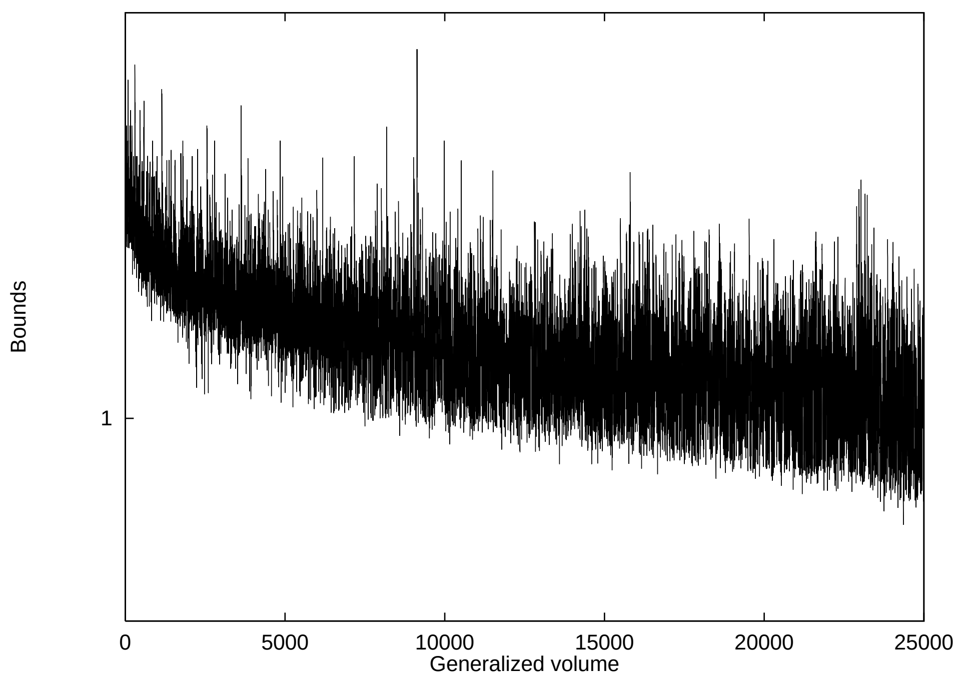


Figure 30: $\prod_{i=1}^{n}(x_i^U - x_i^L)$ versus iteration number for problem fp_3_1, version 2

Variable	Widest Splits[a]	Best Eliminate[b]	Best Point[c]	Best Bisect[d]
1	0	232	1204	245
2	0	548	1846	403
3	0	364	2628	422
4	1320	51	14	11
5	1001	5	29	0
6	826	7	241	17
7	803	32	515	0
8	811	1	27	0

[a]Chosen when the widest quadratic variable is 100 or more times the width of the recommended variable.
[b]The variable with the highest sensitivity split at a location to eliminate one of the regions.
[c]The variable with the highest sensitivity split at the current point.
[d]The variable with the highest sensitivity bisected.

Table 15: Branch types for problem fp_3_1, version 2

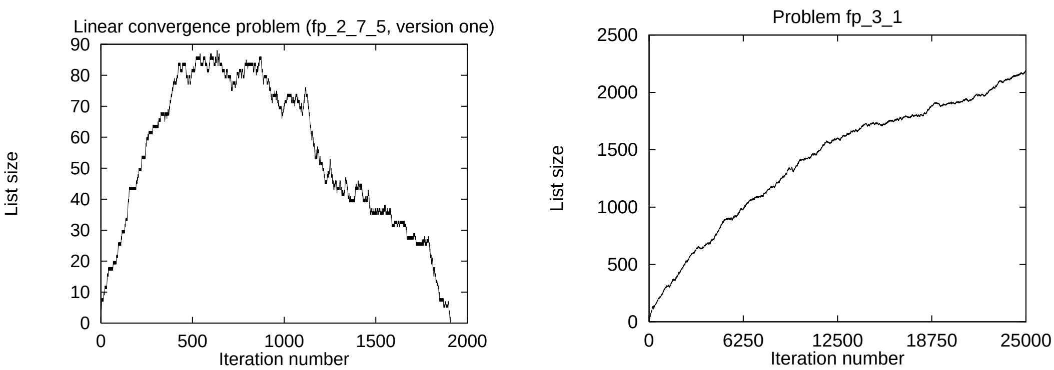


Figure 31: Comparison of list size behavior

keeps growing. Figure 31 compares the list size behavior for fp_3_1 with the typical behavior shown by version 1 applied to problem fp_2_7_5.

The Newton constraint turned out to be insignificant on many of the problems. For version 1, the constraint was only active in problems fp_3_1, fp_4_9, and e_1. For version 2, it was only active in problems fp_3_1, fp_3_4, fp_4_7, fp_4_9, f_a, f_f, s_1d, and s_3. In the other 41 problems, it was never active in the solution of a covering program. Rerunning these problems with the Newton constraint removed

	Version 2 Without Newton		Version 2 With Newton	
	Time (s)	Regions	Time (s)	Regions
fp_3_1	4644.20	25000	4837.56	25000
fp_3_4	641.06	7129	595.69	6269
fp_4_7	0.32	17	0.37	17
fp_4_9	21639.35	25000	19472.80	25000
e_1	0.11	7	0.12	7
f_a	0.52	23	0.92	23
f_f	0.47	11	0.27	9
s_1d	0.24	13	0.26	13
s_3	0.13	3	0.18	3

Table 16: Effect of Newton constraint on the performance of version 2

from the covering program had a small effect on the overall performance of the algorithm, as shown in Table 16.

7.2 The Parallel Algorithm

The parallel algorithm was developed in C++ on a DECstation 3100 and a NeXTstation Turbo using the Message-Passing Interface (MPI) [31] library developed at Argonne National Laboratory, MPICH version 1.0.8. Preliminary testing of the communications protocol took place on the DECstation 3100 and NeXTstation, and later testing took place on a network of seven HP 9000/735 workstations at the Model Advanced Facility (MAF), University of Wisconsin-Madison. Final testing and timing took place on the MAF's IBM SP-2 parallel computer.

Table 17 shows the configuration of the IBM SP-2. It is worth pointing out that eight of the sixteen processors run at a clock speed 5MHz above the remaining eight processors, and four of the fast processors have 384MB of additional RAM

Name	RAM	Clock speed (MHz)
spa01	128MB	62
spa02	128MB	62
spa03	128MB	67
spa04	128MB	67
spa05	128MB	67
spa06	128MB	67
spa07	512MB	67
spa08	512MB	67
spa09	512MB	67
spa10	512MB	67
spa11	128MB	62
spa12	128MB	62
spa13	128MB	62
spa14	128MB	62
spa15	128MB	62
spa16	128MB	62

Table 17: IBM SP-2 configuration

as compared to the other twelve processors. Parallel jobs on the machine are managed by IBM's Load Leveler which pseudo-randomly assigns jobs to processors, so the data reflects runs on processors with different characteristics. The sixteen processors are linked by the SP2 communication subsystem which uses a High-Performance Swith to achieve low latency and high bandwidth.

Tables 18–20 show the runtimes and number of regions examined for all of the problems whose sequential version requires more than one region to be examined. All times presented are wall clock time because that is the best measure for a parallel algorithm. It is pointless to use the parallel algorithm on problems requiring only one region because there is nothing to parallelize. All of these runs use version 2 of the region analysis procedure, the iterative LP.

One of the chief measures of effectiveness for a parallel algorithm is speedup,

Name	1 Node Time (s) Regions	2 Nodes Time (s) Regions	4 Nodes Time (s) Regions	8 Nodes Time (s) Regions	16 Nodes Time (s) Regions
fp_2_1	1.08 39	0.59 36	0.56 51	0.33 45	0.31 51
fp_2_3	0.11 3	0.12 4	0.12 5	0.12 5	0.14 5
fp_2_4	0.10 7	0.12 7	0.09 7	0.10 7	1.22 15
fp_2_5	0.58 21	0.41 21	0.41 20	0.46 21	0.49 19
fp_2_6	1.58 27	1.18 27	1.46 51	1.23 63	1.04 88
fp_2_7_1	240.64 751	122.17 672	200.63 1070	39.10 805	37.94 1001
fp_2_7_2	238.22 461	127.14 520	275.27 996	49.74 896	47.70 1284
fp_2_7_3	800.84 941	788.67 2111	1823.24 8882	945.42 10153	515.07 11435
fp_2_7_4	215.04 595	70.92 563	143.35 1067	27.76 625	42.02 1134
fp_2_7_5	629.90 1981	7823.41 27053	3496.61 26494	1656.58 28318	732.05 25039
fp_2_8	100.28 117	78.75 176	40.06 164	32.91 270	33.89 398
fp_3_1	4436.23 25000×	3790.21 49717×	4628.62 80290×	3295.38 110479	1592.15 111085
fp_3_2	10.98 61	4.70 45	4.63 81	8.12 293	2.16 127
fp_3_3	0.29 7	0.26 11	0.52 17	0.47 33	0.55 45
fp_3_4	463.45 7025	292.50 6868	156.70 6633	65.52 6115	34.45 6361
fp_4_3	0.15 5	0.11 7	0.47 121	0.32 87	0.35 47

× after the region number indicates failure.

Table 18: Runtimes for 1, 2, 4, 8, and 16 processors, part I

Name	1 Node Time (s) Regions	2 Nodes Time (s) Regions	4 Nodes Time (s) Regions	8 Nodes Time (s) Regions	16 Nodes Time (s) Regions
fp_4_4	0.19	0.28	0.26	0.20	0.22
	17	31	42	47	45
fp_4_5	0.75	0.32	0.17	0.16	0.13
	33	37	27	27	27
fp_4_6	0.23	0.13	0.10	0.11	1.12
	11	16	13	13	19
fp_4_7	0.40	0.21	0.20	1.43	2.41
	17	9	19	19	21
fp_4_9	20449.17	n/a	14918.41	13779.19	14016.26
	25000×	n/a	88337×	168781×	328248×
f_a	0.25	0.32	0.19	0.19	0.48
	27	27	13	13	13
f_d	51.80	17.60	17.79	11.59	12.86
	219	181	330	178	325
f_e	0.18	0.17	0.23	0.22	0.22
	3	3	3	3	3
f_f	0.31	0.17	0.15	0.49	0.55
	9	11	11	11	11
f_g_I	0.30	0.30	0.24	0.51	0.56
	3	3	3	3	3
f_g_II	0.29	0.09	0.14	0.12	0.16
	3	3	3	3	3
f_g_III	0.94	0.70	0.65	0.97	0.96
	7	15	27	40	41
f_wingo	0.23	0.15	0.34	0.17	0.17
	27	23	27	27	27

× after the region number indicates failure.

Table 19: Runtimes for 1, 2, 4, 8, and 16 processors, part II

Name	1 Node Time (s) Regions	2 Nodes Time (s) Regions	4 Nodes Time (s) Regions	8 Nodes Time (s) Regions	16 Nodes Time (s) Regions
s_1	0.28	0.18	0.15	0.16	0.17
	19	19	19	21	19
s_1b	0.32	0.25	0.18	0.20	0.20
	21	21	21	21	21
s_1c	0.11	0.08	0.47	0.07	0.30
	7	7	7	7	7
s_1d	0.26	0.18	0.15	0.13	0.23
	13	13	13	13	13
s_2	0.14	0.06	0.03	0.06	0.06
	5	5	5	5	5
s_2b	0.20	0.26	0.11	0.19	0.18
	13	13	9	13	13
s_3	0.12	0.15	0.13	0.13	0.11
	3	3	3	3	3
s_4	2.36	2.26	2.53	0.39	0.85
	38	233	435	55	288
s_5	0.16	0.09	0.09	0.11	0.09
	17	17	19	19	19
s_6	0.29	0.30	0.23	0.22	0.22
	3	3	3	3	3
e_1	0.11	0.10	0.66	0.11	0.11
	7	7	7	7	7
w	32.43	98.04	66.02	93.32	60.62
	203	1099	2361	7205	10261
w_b	12.07	6.96	20.46	13.63	7.33
	105	135	1184	1589	1710
w_c	11.32	9.37	10.89	12.74	8.84
	112	399	1076	2264	3234

Table 20: Runtimes for 1, 2, 4, 8, and 16 processors, part III

which is defined as the ratio of the time required for the fastest sequential algorithm over the time required for the parallel program. In this case as in many, it is not clear what the fastest sequential algorithm is for each particular problem, so the speedup presented here is the ratio of the time it takes for the algorithm running on one processor over the time for p processors. The communications and load balancing routines require a negligible amount of time when running on one processor.

For the sake of presenting the speedup data, the problem are divided into two categories: hard and trivial. Hard problems are those that took at least a second to run on a single processor, and trivial problem took less than a second. Speedup for the hard problems is presented in Table 21; Table 22 shows the results for the trivial problems.

Regarding Table 21, the data for problems fp_3_1 and fp_4_9 requires interpretation. The algorithm was not able to solve fp_4_9, and it took 8 or 16 processors to solve fp_3_1. The time for runs which failed is taken as the time it took for one of the processors to reach a limit of 25000 regions. It is interesting to note that a problem that was unsolvable with 1 processor was solvable with 8.

The results for the other hard problems are somewhat erratic. Problem fp_3_4 shows the best improvement, with consistent improvement as the number of processors increase. In a number of cases the algorithm demonstrates the detrimental anomaly for parallel branch and bound. In some cases, the algorithm is considering far more regions than the single processor version, and it sometimes requires more time than the sequential version (when the speedup is less than 1). In these cases, the algorithm is expanding parts of the tree that the sequential algorithm did

	Number of Processors				
Name	1	2	4	8	16
fp_2_1	1.0000	1.8305	1.9286	3.2727	3.4839
fp_2_6	1.0000	1.3390	1.0822	1.2846	1.5192
fp_2_7_1	1.0000	1.9697	1.1994	6.1545	6.3426
fp_2_7_2	1.0000	1.8737	0.8654	4.7893	4.9941
fp_2_7_3	1.0000	1.0154	0.4392	0.8471	1.5548
fp_2_7_4	1.0000	3.0321	1.5001	7.7464	5.1176
fp_2_7_5	1.0000	0.0805	0.1801	0.3802	0.8605
fp_2_8	1.0000	1.2734	2.5032	3.0471	2.9590
fp_3_1	1.0000×	1.1704×	0.9584×	1.3462	2.7863
fp_3_2	1.0000	2.3362	2.3715	1.3522	5.0833
fp_3_4	1.0000	1.5844	2.9576	7.0734	13.4528
fp_4_9	1.0000×	n/a×	1.3707×	1.4841×	1.4590×
f_d	1.0000	2.9432	2.9117	4.4694	4.0280
s_4	1.0000	1.0442	0.9328	6.0513	2.7765
w	1.0000	0.3308	0.4912	0.3475	0.5350
w_b	1.0000	1.7342	0.5899	0.8855	1.6467
w_c	1.0000	1.2081	1.0395	0.8885	1.2805

× after the region number indicates failure.

Table 21: Speedup for hard problems (1, 2, 4, 8, and 16 processors)

not have to consider.

The speedup for the trivial problems is erratic as well. Many of these problems require only a few regions, and so there is not much work to be shared among the processors. The algorithm also starts out with one processor with all of the work, and work for the other processors only becomes available as that node splits and distributes work. One alternative algorithm was considered which split the most nonlinear variable into p regions, and each of the processors was given a region to start with. That alternative was inferior to the one presented.

	Number of Processors				
Name	1	2	4	8	16
fp_2_3	1.0000	0.9167	0.9167	0.9167	0.7857
fp_2_4	1.0000	0.8333	1.1111	1.0000	0.0820
fp_2_5	1.0000	1.4146	1.4146	1.2609	1.1837
fp_3_3	1.0000	1.1154	0.5577	0.6170	0.5273
fp_4_3	1.0000	1.3636	0.3191	0.4688	0.4286
fp_4_4	1.0000	0.6786	0.7308	0.9500	0.8636
fp_4_5	1.0000	2.3438	4.4118	4.6875	5.7692
fp_4_6	1.0000	1.7692	2.3000	2.0909	0.2054
fp_4_7	1.0000	1.9048	2.0000	0.2797	0.1660
f_a	1.0000	0.7812	1.3158	1.3158	0.5208
f_e	1.0000	1.0588	0.7826	0.8182	0.8182
f_f	1.0000	1.8235	2.0667	0.6327	0.5636
f_g_I	1.0000	1.0000	1.2500	0.5882	0.5357
f_g_II	1.0000	3.2222	2.0714	2.4167	1.8125
f_g_III	1.0000	1.3429	1.4462	0.9691	0.9792
f_wingo	1.0000	1.5333	0.6765	1.3529	1.3529
s_1	1.0000	1.5556	1.8667	1.7500	1.6471
s_1b	1.0000	1.2800	1.7778	1.6000	1.6000
s_1c	1.0000	1.3750	0.2340	1.5714	0.3667
s_1d	1.0000	1.4444	1.7333	2.0000	1.1304
s_2	1.0000	2.3333	4.6667	2.3333	2.3333
s_2b	1.0000	0.7692	1.8182	1.0526	1.1111
s_3	1.0000	0.8000	0.9231	0.9231	1.0909
s_5	1.0000	1.7778	1.7778	1.4545	1.7778
s_6	1.0000	0.9667	1.2609	1.3182	1.3182
e_1	1.0000	1.1000	0.1667	1.0000	1.0000

Table 22: Speedup for trivial problems (1, 2, 4, 8, and 16 processors)

Chapter 8

Conclusions

8.1 Bounding Program

Chapter 4 presents the covering program as a bounding method for the wide class of NLPs in factorable form. It is capable of producing the exact lower bound for a region of finite size when constructed at the region's global minimizer, and so can provide finite termination of branch and bound for nonconvex NLPs without requiring 2^n subproblems.

The size of the covering program is a linear function of the number of variables, number of original constraints, and the number of unique nonlinear terms appearing quadratically or in single variable functions. For dense problems, the number of quadratic terms is proportional to n^2, so the covering program could become too large for conventional solvers. However, the majority of engineering problems will be sparse, and for these the number of additional constraints grows linearly with n according to the test problem set.

A method of treating linear equations with interval coefficients with a system of linear inequalities was also developed. Experience with these systems shows that the supplementary constraints presented significantly improve the tightness of the bounds obtained.

8.2 Branch and Bound Algorithm

Both versions of branch and bound algorithm have been shown to be successful at solving to a high tolerance a variety of problems including concave and indefinite quadratic programs, bilinear programs, polynomial programs, and quadratic NLPs with nonlinear transcendental functions. The time required for solution is highly program dependent, but correlates somewhat with the number of quadratic terms.

Both versions of the algorithm perform about equally in terms of runtime, but version 2 of the algorithm requires far fewer constraint and constraint gradient evaluations. The evaluations required for version 1 might be reduced by using a different local NLP solver such as successive quadratic programming. MINOS was used primarily because of its availability and its ability to solve both NLPs and LPs efficiently.

Version 1 has an advantage in some problems because MINOS is able to locate the global minimum faster and more accurately than the iterative LP of version 2. In comparison, MINOS solved many more fast subproblems, while version 2 solves fewer slow subproblems. Version 1 often has a better feasible point and more accurate Lagrange multipliers to use. Version 2 is sometimes able to provide better bounds on a region because it typically evaluates several covering programs in each region, and the lower bound for the region is the maximum of all of these programs. Version 1 only evaluates one covering program per region. Having better bounds can result in faster termination.

In the cases where the algorithm fails or performs poorly, the primary cause of the poor performance is the branching rules. The problems with poor runtimes use a higher percentage of widest-variable splits. In some cases, the widest split is needed because the sensitivity analysis ignores the effect of some variables' bounds, and in other cases, the widest split rule is a hindrance to success because it causes the algorithm to split on variables that do not matter. The majority of the problems are solved using the best split over 95% of the time.

8.3 Parallel Branch and Bound

The parallel branch and bound algorithm gave erratic results. Some problems show consistent speedup for 2, 4, 8, and 16 processors, and for other problems the time for solving with one processor is less than the time for 2, 4, 8, or 16 processors. The primary reason for the poor speedup when it occurs is that the parallel branch and bound algorithm ends up examining many more regions than the single processor version. The additional work more than cancels out the benefit of having additional processing capability.

The parallel branch and bound algorithm produces more regions because the processors are not always evaluating the most important regions at any particular time. In this case, the processors should be working on the regions with the least lower bound. Because each region keeps its own separate list of regions, some processors are working on regions with higher lower bounds than others. The strategy of sending the second priority region to the right neighbor every 4–7 iterations was inadequate for dispersing the most important work. The high priority work will be dispersed more rapidly if high priority regions are sent more frequently at the cost of increased communications. Another possible improvement is to have each node cycle through the list of nodes in determining who to send its high priority region to. Under the current scheme, high priority regions may have to make their way around the ring before each node is working on high priority work.

8.4 Recommendations for Future Work

Experience with the bounding programs and algorithms presented here leads to the following recommendations for future research:

- Use a more sophisticated treatment of the single variable nonlinear functions. Some single variable nonlinear functions have linear convex or concave envelopes which could be included in the bounding program in addition to the quadratic underestimation.

- Use a reliable implementation of sequential quadratic programming (SQP) instead of MINOS 5.4 for solving the NLPs in version 1. MINOS 5.4 is fairly reliable, but SQP may be faster in many cases.

- Use a hybrid of region analysis procedures version 1 and 2. Start the region analysis procedure with the solution of the covering program in case the region can be pruned, and then use SQP until its line search jams or it finds a local minimum. In the case of a line search jam, use the solution of another covering program to get a new search direction and then resume SQP. In the case of a local minimum, solve the covering program at that point and terminate the region analysis loop.

- Experiment with different load balancing schemes for the parallel algorithm.

Bibliography

[1] A. Aggarwal and C. A. Floudas. A decomposition approach for global optimum search in QP, NLP and MINLP problems. *Annals of Operations Research*, 25:119–146, 1990.

[2] A. Aggarwal and C. A. Floudas. Synthesis of general distillation sequences — Nonsharp separation. *Computers and Chemical Engineering*, 14, 1990.

[3] F. A. Al-Khayyal. Global optimization of concave functions subject to quadratic constraints: An application in nonlinear bilevel programming. *Annals of Operations Research*, 34:125–147, 1992.

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

[5] F. A. Al-Khayyal and C. Larsen. Global optimization of a quadratic function subject to a bounded mixed integer constraint set. *Annals of Operations Research*, 25:169–180, 1990.

[6] E. Balas. Intersection cuts — A new type of cutting planes for integer programming. *Operations Research*, 19:19–39, 1971.

[7] E. Balas. Integer programming and convex analysis: Intersection cuts from outer polars. *Mathematical Programming*, 2:330–382, 1972.

[8] E. Balas. Nonconvex quadratic programming via generalized polars. *SIAM Journal of Applied Mathematics*, 28(2):335–349, March 1975.

[9] E. Balas, V. J. Bowman, F. Glover, and D. Sommer. An intersection cut from the dual of the unit hypercube. *Operations Research*, 19:40–44, 1971.

[10] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, Inc., second edition, 1993.

[11] E. M. L. Beale and J. A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In John Lawrence, editor, *Proceeding of the Fifth International Conference on Operational Research*. Tavistock Publications, 1970.

[12] H. P. Benson. On the convergence of two branch-and-bound algorithms for nonconvex programming problems. *Journal of Optimization Theory and Applications*, 36(1), January 1982. Technical Note.

[13] H. P. Benson and S. Sayin. A finite concave minimization algorithm using branch and bound and neighbor generation. *Journal of Global Optimization*, 5:1–14, 1994.

[14] E. W. Cheney and A. A. Goldstein. Newton's method for convex programming and tchebycheff approximation. *Numerische Mathematik*, 1:253–268, 1959.

[15] J. E. Cope and B. W. Rust. Bounds on solutions of linear systems with inaccurate data. *SIAM Journal on Numerical Analysis*, 16(6):950–963, December 1979.

[16] R. W. Cottle and W. C. Mylander. Ritter's cutting plane method for nonconvex quadratic programming. In J. Abadie, editor, *Integer and Nonlinear Programming*, chapter 11, pages 257–283. North-Holland Publishing Company, 1970.

[17] A. Deif. *Sensitivity Analysis in Linear Systems*. Springer-Verlag, 1986.

[18] W. Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, 1967.

[19] J. E. Falk and K. R. Hoffman. A successive underestimation method for concave minimization problems. *Mathematics of Operations Research*, 1(3):251–259, August 1976.

[20] J. E. Falk and R. M. Soland. An algorithm for separable nonconvex programming problems. *Management Science*, 15(9):550–569, May 1969.

[21] C. A. Floudas, A. Aggarwal, and A. R. Ciric. Global optimum search for nonconvex NLP and MINLP problems. *Computers and Chemical Engineering*, 13(10):1117–1132, 1989.

[22] C. A. Floudas and P. M. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*, volume 455 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[23] C. A. Floudas and P. M. Pardalos, editors. *Recent Advances in Global Optimization.* Princeton Series in Computer Science. Princeton University Press, 1992.

[24] C. A. Floudas and V. Visweswaran. A global optimization algorithm (GOP) for certain classes of nonconvex NLPs — I. Theory. *Computers and Chemical Engineering*, 14(12):1397–1417, 1990.

[25] T. Gal. *Postoptimal Analyses, Parametric Programming, and Related Topics.* McGraw-Hill Inc., 1979.

[26] E. A. Galperin. The cubic algorithm. *Journal of Mathematical Analysis and Applications*, 112:635–640, 1985.

[27] E. A. Galperin and Q. Zheng. Nonlinear observation via global optimization methods. In *Proceedings of the 1983 Conference on Information Sciences and Systems*, pages 620–628, Baltimore, Maryland, 1983.

[28] M. R. Garey and D. S. Johnson. *Computer and Intractability.* W. H. Freeman and Company, 1979.

[29] A. M Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[30] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Maintaining LU factors of a general sparse matrix. *Linear Algebra and Its Applications*, 88/89:239–270, 1987.

[31] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* Scientific and Engineering Computation. MIT Press, 1994.

[32] A. S. E. Hamed and G. P. McCormick. Calculation of bounds on variables satisfying nonlinear inequality constraints. *Journal of Global Optimization*, 3:25–47, 1993.

[33] E. R. Hansen. *Global Optimization using Interval Analysis.* Marcel Dekker, 1992.

[34] P. Hansen, B. Jaumard, and J. Xiong. Decomposition and interval arithmetic applied to global minimization of polynomial and rational functions. *Journal of Global Optimization*, 3:431–437, 1993.

[35] K. L. Hoffman. A method for globally minimizing concave functions over convex sets. *Mathematical Programming*, 20:22–32, 1981.

[36] R. Horst. An algorithm for nonconvex programming. *Mathematical Programming*, 10:312–321, 1976.

[37] R. Horst. A note on the convergence of an algorithm for nonconvex programming problems. *Mathematical Programming*, pages 237–238, 1980. Short Communication.

[38] R. Horst. A general class of branch-and-bound methods in global optimization with some new approaches for concave minimization. *Journal of Optimization Theory and Applications*, 51(2):271–291, November 1986.

[39] R. Horst. Deterministic global optimization with partition sets whose feasibility is not known: Application to concave minimization, reverse convex constraints, DC-programming, and Lipschitzian optimization. *Journal of Optimization Theory and Applications*, 58(1):11–37, July 1988.

[40] R. Horst. Deterministic methods in constrained global optimization: Some recent advances and new fields of application. *Naval Research Logistics Quarterly*, 37:433–471, 1990.

[41] R. Horst and J. de Vries. On finding new vertices and redundant constraints in cutting plane algorithms for global optimization. *Operations Research Letters*, 7(2):85–90, April 1988.

[42] R. Horst and N. V. Thoai. Branch-and-bound methods for solving systems of Lipschitzian equations and inequalities. *Journal of Optimization Theory and Applications*, 58(1):139–145, July 1988. Technical Note.

[43] R. Horst and N. V. Thoai. Conical algorithm for the global minimization of linearly constrained decomposable concave minimization problems. *Journal of Optimization Theory and Applications*, 74(3):469–486, September 1992.

[44] R. Horst and N. V. Thoai. Constraint decomposition algorithms in global optimization. *Journal of Global Optimization*, 5:333–348, 1994.

[45] R. Horst, N. V. Thoai, and H. P. Benson. Concave minimization via conical partitions and polyhedral outer approximation. *Mathematical Programming*, 50:259–274, 1991.

[46] R. Horst, N. V. Thoai, and J. de Vries. A new simplicial cover technique in constrained global optimization. *Journal of Global Optimization*, 2:1–19, 1992.

[47] R. Horst, N. V. Thoai, and J. de Vries. On geometry and convergence of a class of simplicial covers. *Optimization*, 25:53–64, 1992.

[48] R. Horst and H. Tuy. On the convergence of global methods in multiextremal optimization. *Journal of Optimization Theory and Applications*, 54(2):253–271, August 1987.

[49] R. Horst and H. Tuy. *Global Optimization, Deterministic Approaches.* Springer-Verlag, 1990.

[50] D. L. Miller J. F. Pekny and G. J. McRae. An exact algorithm for scheduling when production costs depend on consecutive system states. *Computers and Chemical Engineering*, 14(9):1009–1023, 1990.

[51] R. B. Kearfott. Interval newton/generalized bisection when there are singularities near roots. *Annals of Operations Research*, 25:181–196, 1990.

[52] R. B. Kearfott. An interval branch and bound algorithm for bound constrained optimization problems. *Journal of Global Optimization*, 2:259–280, 1992.

[53] J. E. Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, December 1960.

[54] G. R. Kocis and I. E. Grossman. Global optimization of nonconvex mixed-integer nonlinear programming (MINLP) problems in process synthesis. *Industrial Engineering and Chemical Research*, 27:1407–1421, 1988.

[55] G. R. Kocis and I. E. Grossmann. Relaxation strategy for the structural optimization of process flow sheets. *I&EC Research*, 26(9):1869, 1987.

[56] A. C. Kokossis and C. A. Floudas. Optimization of complex reactor networks — I. isothermal operation. *Computers and Chemical Engineering*, 45(3):595–614, 1990.

[57] H. Konno. A cutting plane algorithm for solving bilinear programs. *Mathematical Programming*, 11:14–27, 1976.

[58] H. Konno. Maximization of a convex quadratic function under linear constraints. *Mathematical Programming*, 11:117–127, 1976.

[59] H. Konno and T. Kuno. Generalized linear multiplicative and fractional programming. *Annals of Operations Research*, 25:147–162, 1990.

[60] P. F. Kough. The indefinite quadratic programming problem. *Operations Research*, 27(3):516–533, May–June 1979.

[61] T. Lai and S. Sahni. Anomalies in parallel branch-and-bound algorithms. *Communications of the ACM*, 27(6):594–602, 1984.

[62] W. B. Liu and C. A. Floudas. A remark on the GOP algorithm for global optimization. *Journal of Global Optimization*, 3:519–521, 1993.

[63] W. B. Liu and C. A. Floudas. Convergence of the (gop) algorithm for a large class of smooth optimization problems. *Journal of Global Optimization*, 6:207–211, 1995.

[64] G. P. McCormick. Attempts to calculate global solutions of problems that may have local minima. In F. A. Lootsma, editor, *Numerical methods for nonlinear optimization*, pages 209–221. University of Dundee, Academic Press, 1972.

[65] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.

[66] G. P. McCormick. *Nonlinear Programming, Theory, Algorithms, and Applications*. John Wiley & Sons, 1983.

[67] C. M. McDonald and C. A. Floudas. Decomposition based and branch and bound global optimization approaches for the phase equilibrium problem. *Journal of Global Optimization*, 5:205–251, 1994.

[68] G. P. McKeown, V.J. Rayward-Smith, and S. A. Rush. Parallel branch-and-bound. In *Advances in Parallel Algorithms*, chapter 5, pages 111–150. Halsted Press, 1992.

[69] A. Mfayokurera. Nonconvex phase equilibria computations by global minimization. Master's thesis, University of Wisconsin-Madison, June 1989.

[70] R. H. Mladineo. An algorithm for finding the global maximum of a multimodal, multivariate function. presented at the 12th International Symposium on Mathematical Programming, Cambridge, Mass., 1985.

[71] R. E. Moore. Global optimization to prescribed accuracy. *Computers and Mathematics with Applications*, 21(6/7):25–39, 1991.

[72] R. E. Moore and H. Ratschek. Inclusion functions and global optimization II. *Mathematical Programming*, 41:341–356, 1988.

[73] B. A. Murtagh and M. A. Saunders. A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. *Mathematical Programming Study*, 16:84–117, 1982.

[74] B. A. Murtagh and M. A. Saunders. MINOS 5.1 user's guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Stanford University, Stanford, CA 94305-4022, January 1987.

[75] L. D. Muu and W. Oettli. Combined branch-and-bound and cutting plane methods for solving a class of nonlinear programming problems. *Journal of Global Optimization*, 3:377–391, 1993.

[76] A. Neumaier. *Interval Methods for Systems of Equations*. Encyclopedia of Mathematics and Its Applications. Cambridge University Press, 1990.

[77] W. Oettli. On the solution set of a linear system with inaccurate coefficients. *SIAM Journal on Numerical Analysis*, 2(1):115–118, 1965.

[78] W. Oettli, W. Prager, and J. H. Wilkinson. Admissible solutions of linear systems with not sharply defined coefficients. *SIAM Journal on Numerical Analysis*, 2(2):291–299, 1965.

[79] P. M. Pardalos. Parallel search algorithms in global optimization. *Applied Mathematics and Computation*, 29:219–229, 1989.

[80] P. M. Pardalos. Global optimization algorithms for linearly constrained indefinite quadratic problems. *Computers and Mathematics with Applications*, 21(6/7):87–97, 1991.

[81] P. M. Pardalos, J. H. Glick, and J. B. Rosen. Global minimization of indefinite quadratic problems. *Computing*, 39:281–291, 1987.

[82] P. M. Pardalos and A. T. Phillips. Global optimization of fractional programs. *Journal of Global Optimization*, 1:173–182, 1991.

[83] P. M. Pardalos and J. B. Rosen. Methods for global concave minimization: A bibliographic survey. *SIAM Review*, 28(3):367–379, September 1986.

[84] P. M. Pardalos and J. B. Rosen. *Constrained Global Optimization: Algorithms and Applications*, volume 268 of *Lecture Notes in Computer Science*. Springer-Verlag, 1987.

[85] P. M. Pardalos and G. Schnitger. Checking local optimality in constrained quadratic programming is NP-hard. *Operations Research Letters*, 7(1):33–35, February 1988.

[86] P. M. Pardalos and S. A. Vavasis. Quadratic programming with one negative eigenvalue is NP-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

[87] A. T. Phillips and J. B. Rosen. A parallel algorithm for constrained concave quadratic global minimization. *Journal of Optimization Theory and Applications*, 42:421–448, 1988.

[88] A. T. Phillips and J. B. Rosen. Guaranteed ϵ-approximate solution for indefinite quadratic global minimization. *Naval Research Logistics*, 37:499–514, 1990.

[89] A. T. Phillips and J. B. Rosen. A parallel algorithm for partially separable non-convex global minimization: Linear constraints. *Annals of Operations Research*, 25:101–118, 1990.

[90] A. T. Phillips and J. B. Rosen. Computational comparison of two methods for constrained global optimization. *Journal of Global Optimization*, pages 325–332, 1994.

[91] T. Q. Phong, P. D. Tao, and L. T. H. An. A method for solving d.c. programming problems. application to fuel mixture nonconvex optimization problem. *Journal of Global Optimization*, 6:87–105, 1995.

[92] S. A. Pijavskii. An algorithm for finding the absolute extremum of a function. *USSR Computational Mathematics and Mathematical Physics*, 12:57–67, 1972.

[93] J. Pinter. A unified approach to globally convergent one-dimensional optimization algorithms. Technical Report IAMI-83.5, Istituto per la Applicazioni della Matematica e dell'Informatica, Milan, Italy, 1983.

[94] J. Pinter. Globally convergent methods for n-dimensional multiextremal optimization. *Optimization*, 17:1–16, 1986.

[95] I. Quesada and I. E. Grossmann. Global optimization algorithm for heat exchanger networks. *Industrial Engineering and Chemical Research*, 32:487–499, 1993.

[96] I. Quesada and I. E. Grossmann. A global optimization algorithm for linear fractional and bilinear programs. *Journal of Global Optimization*, 6:39–76, 1995.

[97] H. Ratschek. Inclusion functions and global optimization. *Mathematical Programming*, 33:300–317, 1985.

[98] H. Ratschek and J. Rokne. *New Computer Methods For Global Optimization.* Mathematics and its Applications. Ellis Horwood Limited, 1988.

[99] H. Ratschek and R. L. Voller. What can interval analysis do for global optimization? *Journal of Global Optimization*, 1:111–130, 1991.

[100] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell. *Engineering Optimization: Methods and Applications.* John Wiley & Sons, Inc., 1983.

[101] J. B. Rosen. Global minimization of a linearly constrained concave function by partition of feasible domain. *Mathematics of Operations Research*, 8(2):215–230, May 1983.

[102] J. B. Rosen. Performance of approximate algorithms for global minimization. *Mathematical Programming Study*, 22:231–236, 1984.

[103] J. B. Rosen and P. M. Pardalos. Global minimization of large-scale constrained concave quadratic problems by separable programming. *Mathematical Programming*, 34:163–174, 1986.

[104] R. A. Rushmeier and G. L. Nemhauser. Experiments with parallel branch-and-bound algorithms for the set covering problem. *Operations Research Letters*, 13:277–285, 1993.

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

[106] N. V. Sahinidis and I. E. Grossmann. Convergence properties of generalized Benders decomposition. *Computers and Chemical Engineering*, 7:481–491, 1991.

[107] R. B. Schnabel and E. Eskow. A new modified Cholesky factorization. *SIAM Journal on Scientific and Statistical Computing*, 11(6):1136–1158, November 1990.

[108] F. Schoen. Stochastic techniques for global optimization: A survey of recent advances. *Journal of Global Optimization*, 1:207–228, 1991.

[109] H. D. Sherali and A. Alameddine. A new reformulation-linearization technique for bilinear programming problems. *Journal of Global Optimization*, 2:379–410, 1992.

[110] H. D. Sherali and C. M. Shetty. A finitely convergent algorithm for bilinear programming problems using polar cuts and disjunctive face cuts. *Mathematical Programming*, 19:14–31, 1980.

[111] H. D. Sherali and C. H. Tuncbilek. A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Journal of Global Optimization*, 2:101–112, 1992.

[112] B. O. Shubert. A sequential method seeking the global maximum of a function. *SIAM Journal of Numerical Analysis*, 9:379–388, 1972.

[113] R. M. Soland. An algorithm for separable nonconvex programming problems II: Nonconvex constraints. *Management Science*, 17(11):759–773, July 1971.

[114] R. E. Swaney. Global solution of algebraic nonlinear programs. AIChE Annual Meeting (Chicago, IL 1990). Publication pending.

[115] P. T. Thạch and H. Tuy. The relief indicator method for constrained global optimization. *Naval Research Logistics*, 37:473–497, 1990.

[116] N. V. Thoai and H. Tuy. Convergent algorithms for minimizing a concave function. *Mathematics of Operations Research*, 5(4):556–566, November 1980.

[117] A. Törn and A. Ẑilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[118] H. Tuy. Concave programming under linear constraints. *Soviet Mathematics*, pages 1437–1440, 1964.

[119] H. Tuy. Effect of the subdivision strategy on convergence and efficiency of some global optimization algorithms. *Journal of Global Optimization*, 1(1):23–36, 1991.

[120] H. Tuy and R. Horst. Convergence and restart in branch-and-bound algorithms for global optimization. Application to concave minimization and D.C. optimization problems. *Mathematical Programming*, 41:161–183, 1988.

[121] H. Tuy, T. V. Thieu, and Ng. Q. Thai. A conical algorithm for globally minimizing a concave function over a closed convex set. *Mathematics of Operations Research*, 10(3):498–514, August 1985.

[122] R. Vaidyanathan and M. El-Halwagi. Global optimization of nonconvex nonlinear programs via interval analysis. *Computers and Chemical Engineering*, 18(10):889–897, 1994.

[123] H. Vaish and C. M. Shetty. A cutting plane algorithm for the bilinear programming problem. *Naval Research Logistics Quarterly*, 24:83–94, 1977.

[124] V. Visweswaran and C. A. Floudas. A global optimization algorithm (GOP) for certain classes of nonconvex NLPs — II. Application of theory and test problems. *Computers and Chemical Engineering*, 14(12):1419–1434, 1990.

[125] V. Visweswaran and C. A. Floudas. A global optimization algorithm (GOP) for certain classes of nonconvex NLPs — III. New properties for improved computational performance. In *Proceedings of the 4th International Symposium on Process Systems Engineering, Volume 1: Design*, August 1991. Montebello, Quebec, Canada.

[126] V. Visweswaran and C. A. Floudas. Unconstrained and constrained global optimization of polynomial functions in one variable. *Journal of Global Optimization*, 2:73–99, 1992.

[127] V. Visweswaran and C. A. Floudas. New properties and computational improvement of the GOP algorithm for problems with quadratic objective functions and constraints. *Journal of Global Optimization*, 3:439–462, 1993.

[128] R. R. Wehe and A. W. Westerberg. An algorithm procedure for the synthesis of distillation sequences with bypass. *Computers and Chemical Engineering*, 11:619, 1987.

[129] A. W. Westerberg. The synthesis of distillation based separation sequences. *Computers and Chemical Engineering*, 9:421, 1985.

[130] A. W. Westerberg and J. V. Shah. Assuring a global optimum by the use of an upper bound on the lower (dual) bound. *Computers and Chemical Engineering*, 2:83–92, 1978.

[131] Q. Zheng, B. Jiang, and S. Zhuang. A method for finding the global extremum. *Acta Mathematicae Applagatea Sinica*, 2:164–174, 1978. In Chinese.

[132] P. B. Zwart. Computational aspects on the use of cutting planes in global optimization. In *Proceedings of the 1971 annual conference of the ACM*, pages 457–465. ACM, 1971.

Appendix A

Comparison to Swaney's Work

This appendix will show the relationship between the covering program (138) and the covering program given by Swaney [114]. This will be accomplished by showing how Swaney's covering program can be developed from the methods presented in this dissertation.

The first step in the process is to relax the interval LP (138) by dropping constraints (140), (141), (143–146), (152), and (153). The constraint on t with $\mathrm{sign}(\rho_{jk}^{-1}) = \sigma_{jk}$ is dropped, either (147) or (148). Let $\hat{\rho}_{jk}^{-1}$ refer to the value of ρ_{jk}^{-1} with $\mathrm{sign}(\hat{\rho}_{jk}^{-1}) = -\sigma_{jk}$. Lastly, $\sigma_{jk} q_{jk}$ is included in (142) rather than q_{jk}; constraint (142) provides a valid bound for either. With these adjustments, the interval LP becomes

$$\min \quad c^{\mathrm{T}} \Delta x \tag{233}$$

$$\text{s.t.} \quad \bar{b}^{i\mathrm{T}} z + \sum_{j,k} \hat{H}_{jk}^{(i)} (t_{jk} + q_{jk}) \leq 0 \qquad \forall i \in I_A \tag{234}$$

$$0 \leq [w_k^L, w_k^U] e^{j\mathrm{T}} (I - N(N^{\mathrm{T}} N)^{-1} N^{\mathrm{T}}) z - \sigma_{jk} q_{jk} \leq 0 \quad \forall j, k \tag{235}$$

$$0 \leq 2(\sigma_{jk}t_{jk}) + |\hat{\rho}_{jk}^{-1}| \sum_{r,s} |\bar{\gamma}_{rs}|(\sigma_{rs}t_{rs}) \qquad \forall j,k \tag{236}$$

$$z = (I + [0\ N])\Delta x \tag{237}$$

$$\Delta x^L \leq \Delta x \leq \Delta x^U \tag{238}$$

$$z^L \leq z \leq z^U \tag{239}$$

Constraints (236) are replaced with the following with the addition of a slack variable θ_{jk}.

$$\begin{aligned} \theta_{jk} &= 2(\sigma_{jk}t_{jk}) + |\hat{\rho}_{jk}^{-1}| \sum_{r,s} |\bar{\gamma}_{rs}|(\sigma_{rs}t_{rs}) \quad \forall j,k & (240) \\ 0 &\leq \theta_{jk} & (241) \end{aligned}$$

Equation (240) can then be solved for $\sigma_{jk}t_{jk}$, so $\sigma_{jk}t_{jk}$ can be eliminated. Considering σt, $|\bar{\gamma}|$, $|\hat{\rho}^{-1}|$, and θ as vectors with the bilinear terms ordered identically, equation (240) can be rewritten in matrix form.

$$\left(I + \frac{1}{2}|\hat{\rho}^{-1}||\bar{\gamma}^{\mathrm{T}}|\right)\sigma t = \theta \tag{242}$$

The Sherman-Woodbury-Morrison formula [66, page 70] provides a method for inverting the formula on the left to give

$$\begin{aligned} (\sigma t) &= \left(I - \frac{|\hat{\rho}^{-1}||\bar{\gamma}^{\mathrm{T}}|}{2 + |\bar{\gamma}^{\mathrm{T}}||\hat{\rho}^{-1}|}\right)\theta & (243) \\ (\sigma_{jk}t_{jk}) &= \theta_{jk} - \frac{|\hat{\rho}_{jk}^{-1}| \sum_{r,s} |\bar{\gamma}_{rs}|\theta_{rs}}{2 + \sum_{r,s} |\bar{\gamma}_{rs}||\hat{\rho}_{jk}^{-1}|} & (244) \end{aligned}$$

Substituting in this result and solving (235) for $\sigma_{jk}g_{jk}$ gives

$$\min \quad c^{\mathrm{T}}\Delta x \tag{245}$$

$$\text{s.t.} \quad \bar{b}^{i\mathrm{T}}z + \sum_{j,k} \sigma_{jk}\hat{H}_{jk}^{(i)}[w_k^L, w_k^U]e^{j\mathrm{T}}(I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}})z + \qquad \forall i \in I_A \tag{246}$$
$$\sum_{j,k} \sigma_{jk}\hat{H}_{jk}^{(i)}\theta_{jk} - \sum_{j,k} \sigma_{jk}\hat{H}_{jk}^{(i)} \frac{|\hat{\rho}_{jk}^{-1}| \sum_{r,s} |\bar{\gamma}_{rs}|\theta_{rs}}{2 + \sum_{r,s} |\bar{\gamma}_{rs}||\hat{\rho}_{jk}^{-1}|} \leq 0$$

$$z = (I + [0\ N])\Delta x \tag{247}$$

$$\Delta x^L \leq \Delta x \leq \Delta x^U \tag{248}$$

$$0 \leq \theta \tag{249}$$

$$z^L \leq z \leq z^U \tag{250}$$

Reversing the order of the sums in (246), applying the two definitions in (255,256), and dropping Δx gives

$$\min \quad c^{\mathrm{T}} z \tag{251}$$

$$\text{s.t.} \quad \bar{b}^{i\mathrm{T}} z + \sum_{j,k} \sigma_{jk} \hat{H}^{(i)}_{jk} [w^L_k, w^U_k] e^{j\mathrm{T}} (I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}}) z + \sum_{j,k} \tau^{(i)}_{jk} \theta_{jk} \leq 0 \qquad \forall i \in I_A \tag{252}$$

$$0 \leq \theta \tag{253}$$

$$z^L \leq z \leq z^U \tag{254}$$

where

$$\tau^{(i)}_{jk} = \sigma_{jk} \hat{H}^{(i)}_{jk} - |\bar{\gamma}_{jk}| \xi_i \qquad \forall\, j,k \tag{255}$$

$$\xi_i = \frac{\sum_{r,s} |\hat{\rho}^{-1}_{rs}| \sigma_{rs} \hat{H}^{(i)}_{rs}}{2 + \sum_{r,s} |\hat{\rho}^{-1}_{rs}||\hat{\rho}^{-1}_{rs}|} \qquad i \in I_A \tag{256}$$

The last step before applying the interval LP relaxation is to introduce the definition of interval matrix A.

$$A_{ij} = \bar{b}^i_j + \sum_{r,s} \sigma_{rs} \hat{H}^{(i)}_{rs} [w^L_s, w^U_s] e^{r\mathrm{T}} (I - N(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}}) e^j \tag{257}$$

Introducing A in the LP gives

$$\min \quad c^{\mathrm{T}} z \tag{258}$$

$$\text{s.t.} \quad [\underline{A}, \bar{A}]z + \sum_{j,k} \tau_{jk}^{(i)} \theta_{jk} \leq 0 \quad \forall i \in I_A \tag{259}$$

$$0 \leq \theta \tag{260}$$

$$z^L \leq z \leq z^U \tag{261}$$

and applying the interval LP relaxation provides

$$\min \quad c^{\mathrm{T}}(z^{(+)} - z^{(-)}) \tag{262}$$

$$\text{s.t.} \quad \underline{A}z^{(+)} - \bar{A}z^{(-)} + \sum_{j,k} \tau_{jk}^{(i)} \theta_{jk} \leq 0 \qquad \forall i \in I_A \tag{263}$$

$$|\underline{A}_{i\,s(i)}| z_{s(i)}^{(+)} + |\bar{A}_{i\,s(i)}| z_{s(i)}^{(-)} - \underline{A}^{(-)} z^{(+)} - \bar{A}^{(+)} z^{(-)} - \sum_{j,k} \tau_{jk}^{(i)(-)} \theta_{jk} \leq 0 \qquad \forall i \in I_A \tag{264}$$

$$0 \leq \theta \tag{265}$$

$$0 \leq z^{(+)} \leq z^U \tag{266}$$

$$0 \leq z^{(-)} \leq -z^L \tag{267}$$

These constraints are equivalent to the constraints included in the covering program of Swaney [114].

This dissertation uses a different covering program than Swaney [114] for a number of reasons. In Swaney [114], the covering program is constructed at a local minimum, so it is built around a feasible point with accurate Lagrange multipliers. This dissertation's covering program may be constructed around infeasible points with inaccurate Lagrange multipliers. The significance of the multipliers is that they indicate the direction in which the q_{jk} and t_{jk} variables need to be supported. At a local minimum, one can determine which constraints on q_{jk} are t_{jk} are required and which are not. With an infeasible point and estimated Lagrange multipliers, one cannot be certain which constraints will be required. For these reasons, all of the constraints on the bilinear terms were retained.

Leaving the problem in its original form also makes it easy to introduce new constraints on the quadratic terms. This was demonstrated with the addition of the set of Newton constraints to constraint the null space quadratic terms. This and future extensions are relatively easy with the covering program presented in this dissertation.

Swaney's covering program drops two of the constraints that this dissertation's covering program retains, so the solution to his covering program should be less than or equal to this dissertation's covering program. However, there is some uncertainty because of the interval LP relaxation. This dissertation's covering program contains $\Delta x^{(+)}$, $\Delta x^{(-)}$, $t_{jk}^{(+)}$, $t_{jk}^{(-)}$, $q_{jk}^{(+)}$ and $q_{jk}^{(-)}$ in addition to the plus and minus variables of Swaney's covering program. These additional plus and minus variables can introduce additional slack if the set of constraints (101) is not chosen correctly. The choices made for these constraints are believed to be correct, but they have not been proven so.

Appendix B

Detailed Problem Descriptions

Filename: `fp_2_1.def`

Description: Chapter 2, Test Problem 1 [22], a concave quadratic program

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + 42x_2 + 44x_3 + 45x_4 + 47x_5 + 47.5x_6 + -50x_2x_2 + -50x_3x_3 + \\ & -50x_4x_4 + -50x_5x_5 + -50x_6x_6 \leq 0 \\ & 20x_2 + 12x_3 + 11x_4 + 7x_5 + 4x_6 \leq 40 \\ & -250 \leq x_1 \leq 225.5 \\ & 0 \leq x_2, x_3, x_4, x_5, x_6 \leq 1 \end{aligned} \tag{268}$$

Solution

Objective	-17	x_1	-17	x_2	1
x_3	1	x_4	0	x_5	1
x_6	0				

Filename: `fp_2_2.def`

Description: Chapter 2, Test Problem 2 (with nonrestrictive variable bounds added) [22], a concave quadratic program

$$\min \quad x_1 \tag{269}$$

$$\text{s.t.} \quad -x_1 + -10.5x_2 + -7.5x_3 + -3.5x_4 + -2.5x_5 + -1.5x_6 + -10x_7 +$$

$$-0.5x_2x_2 + -0.5x_3x_3 + -0.5x_4x_4 + -0.5x_5x_5 + -0.5x_6x_6 \leq 0$$

$$6x_2 + 3x_3 + 3x_4 + 2x_5 + x_6 \leq 6.5$$

$$10x_2 + 10x_4 + x_7 \leq 20$$

$$-230 \leq x_1 \leq 0$$

$$0 \leq x_2, x_3, x_4, x_5, x_6 \leq 1$$

$$0 \leq x_7 \leq 20$$

Solution

Objective	-213	x_1	-213	x_2	5.884182031e-15
x_3	1	x_4	0	x_5	1
x_6	1	x_7	20		

Filename: `fp_2_3.def`

Description: Chapter 2, Test Problem 3 [22] (with nonrestrictive variable bounds added), a concave quadratic program

$$\min \quad x_1 \tag{270}$$

$$\text{s.t.} \quad -x_1 + 5x_2 + 5x_3 + 5x_4 + 5x_5 + -x_6 + -x_7 + -x_8 + -x_9 + -x_{10} +$$

$$-x_{11} + -x_{12} + -x_{13} + -x_{14} + -5x_2x_2 + -5x_3x_3 + -5x_4x_4 +$$

$$-5x_5x_5 \leq 0$$

$$2x_2 + 2x_3 + x_{11} + x_{12} \leq 10$$

$$2x_2 + 2x_4 + x_{11} + x_{13} \leq 10$$

$$2x_3 + 2x_4 + x_{12} + x_{13} \leq 10$$

$$-8x_2 + x_{11} \leq 0$$

$$-8x_3 + x_{12} \leq 0$$

$$-8x_4 + x_{13} \leq 0$$

$$-2x_5 + -x_6 + x_{11} \leq 0$$

$$-2x_7 + -x_8 + x_{12} \leq 0$$

$$-2x_9 + -x_{10} + x_{13} \leq 0$$

$$-30 \leq x_1 \leq 5$$

$$0 \leq x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{14} \leq 1$$

$$0 \leq x_{11} \leq 8$$

$$0 \leq x_{12} \leq 8$$

$$0 \leq x_{13} \leq 8$$

Solution

Objective	-15	x_1	-15	x_2	1
x_3	1	x_4	1	x_5	1
x_6	1	x_7	1	x_8	1
x_9	1	x_{10}	1	x_{11}	3
x_{12}	3	x_{13}	3	x_{14}	1

Filename: `fp_2_4.def`

Description: Chapter 2, Test Problem 4 [22] (with nonrestrictive variable bounds added), a concave quadratic program

$$\min \quad x_1 \tag{271}$$

$$\text{s.t.} \quad -x_1 + 6.5x_2 + -x_3 + -2x_4 + -3x_5 + -2x_6 + -x_7 + -0.5x_2x_2 \leq 0$$

$$x_2 + 2x_3 + 8x_4 + x_5 + 3x_6 + 5x_7 \leq 16$$

$$-8x_2 + -4x_3 + -2x_4 + 2x_5 + 4x_6 + -x_7 \leq -1$$

$$2x_2 + 0.5x_3 + 0.2x_4 + -3x_5 + -x_6 + -4x_7 \leq 24$$

$$0.2x_2 + 2x_3 + 0.1x_4 + -4x_5 + 2x_6 + 2x_7 \leq 12$$

$$-0.1x_2 + -0.5x_3 + 2x_4 + 5x_5 + -5x_6 + 3x_7 \leq 3$$

$$-150 \leq x_1 \leq 104$$

$$0 \leq x_2 \leq 16$$

$$0 \leq x_3 \leq 8$$

$$0 \leq x_4, x_7 \leq 2$$

$$0 \leq x_5, x_6 \leq 1$$

Solution

Objective	-11	x_1	-11	x_2	0
x_3	6	x_4	0	x_5	1
x_6	1	x_7	0		

Filename: `fp_2_5.def`

Description: Chapter 2, Test Problem 5 [22], a concave quadratic program

$$\begin{aligned}
\min \quad & x_1 && (272)\\
\text{s.t.} \quad & -x_1 + -20x_2 + -80x_3 + -20x_4 + -50x_5 + -60x_6 + -90x_7 + \\
& 10x_9 + 10x_{10} + 10x_{11} + -5x_2x_2 + -5x_3x_3 + -5x_4x_4 + \\
& -5x_5x_5 + -5x_6x_6 + -5x_7x_7 + -5x_8x_8 \leq 0 \\
& -2x_2 + -6x_3 + -x_4 + -3x_6 + -3x_7 + -2x_8 + -6x_9 + -2x_{10} + \\
& -2x_{11} \leq -4 \\
& 6x_2 + -5x_3 + 8x_4 + -3x_5 + x_7 + 3x_8 + 8x_9 + 9x_{10} + -3x_{11} \leq 22 \\
& -5x_2 + 6x_3 + 5x_4 + 3x_5 + 8x_6 + -8x_7 + 9x_8 + 2x_9 + -9x_{11} \leq -6 \\
& 9x_2 + 5x_3 + -9x_5 + x_6 + -8x_7 + 3x_8 + -9x_9 + -9x_{10} + \\
& -3x_{11} \leq -23
\end{aligned}$$

$-8x_2 + 7x_3 + -4x_4 + -5x_5 + -9x_6 + x_7 + -7x_8 + -x_9 +$

$3x_{10} + -2x_{11} \leq -12$

$-7x_2 + -5x_3 + -2x_4 + -6x_6 + -6x_7 + -7x_8 + -6x_9 +$

$7x_{10} + 7x_{11} \leq -3$

$x_2 + -3x_3 + -3x_4 + -4x_5 + -x_6 + -4x_8 + x_9 + 6x_{10} \leq 1$

$x_2 + -2x_3 + 6x_4 + 9x_5 + 0x_6 + -7x_7 + 9x_8 +$

$-9x_9 + -6x_{10} + 4x_{11} \leq 12$

$-4x_2 + 6x_3 + 7x_4 + 2x_5 + 2x_6 + 0x_7 + 6x_8 +$

$6x_9 + -7x_{10} + 4x_{11} \leq 15$

$x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} \leq 9$

$-x_2 + -x_3 + -x_4 + -x_5 + -x_6 + -x_7 + -x_8 + -x_9 +$

$-x_{10} + -x_{11} \leq -1$

$-360 \leq x_1 \leq 30$

$0 \leq x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} \leq 1$

The value of the objective function reported [22] seems to be in error. Table 23 shows the results of this algorithm.

Objective	-268.0146	x_6	0.7151
x_1	-268.0146	x_7	1.0000
x_2	1.0000	x_8	0.0000
x_3	0.9075	x_9	0.9170
x_4	0.0000	x_{10}	1.0000
x_5	1.0000	x_{11}	1.0000

Table 23: Global minimum of problem fp_2_5

Filename: `fp_2_6.def`

Description: Chapter 2, Test Problem 6 [22], a concave quadratic program

$$\begin{aligned}
\min \quad & x_1 \\
\text{s.t.} \quad & -x_1 + 48x_2 + 42x_3 + 48x_4 + 45x_5 + 44x_6 + 41x_7 + 47x_8 + \\
& 42x_9 + 45x_{10} + 46x_{11} + -50x_2x_2 + -50x_3x_3 + -50x_4x_4 + \\
& -50x_5x_5 + -50x_6x_6 + -50x_7x_7 + -50x_8x_8 + -50x_9x_9 + \\
& -50x_{10}x_{10} + -50x_{11}x_{11} \leq 0 \\
& -2x_2 + -6x_3 + -x_4 + -3x_6 + -3x_7 + -2x_8 + -6x_9 + \\
& -2x_{10} + -2x_{11} \leq -4 \\
& 6x_2 + -5x_3 + 8x_4 + -3x_5 + x_7 + 3x_8 + 8x_9 + \\
& 9x_{10} + -3x_{11} \leq 22 \\
& -5x_2 + 6x_3 + 5x_4 + 3x_5 + 8x_6 + -8x_7 + 9x_8 + 2x_9 + -9x_{11} \leq -6 \\
& 9x_2 + 5x_3 + -9x_5 + x_6 + -8x_7 + 3x_8 + -9x_9 + \\
& -9x_{10} + -3x_{11} \leq -23 \\
& -8x_2 + 7x_3 + -4x_4 + -5x_5 + -9x_6 + x_7 + -7x_8 + \\
& -x_9 + 3x_{10} + -2x_{11} \leq -12 \\
& -500 \leq x_1 \leq 448 \\
& 0 \leq x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} \leq 1
\end{aligned} \tag{273}$$

Solution

Objective	-39	x_1	-39	x_2	1
x_3	0	x_4	0	x_5	1
x_6	1	x_7	1	x_8	0
x_9	1	x_{10}	1	x_{11}	1

Filename: `fp_2_7_1.def`

Description: Chapter 2, Test Problem 7, Case 1 [22] (with nonrestrictive variable bounds added), a concave quadratic program

$$
\begin{aligned}
\min \quad & x_1 & (274)\\
\text{s.t.} \quad & -x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 + 2x_6 + 2x_7 + \\
& 2x_8 + 2x_9 + 2x_{10} + 2x_{11} + 2x_{12} + 2x_{13} + 2x_{14} + \\
& 2x_{15} + 2x_{16} + 2x_{17} + 2x_{18} + 2x_{19} + 2x_{20} + 2x_{21} + \\
& -0.5x_2x_2 + -0.5x_3x_3 + -0.5x_4x_4 + -0.5x_5x_5 + \\
& -0.5x_6x_6 + -0.5x_7x_7 + -0.5x_8x_8 + -0.5x_9x_9 + \\
& -0.5x_{10}x_{10} + -0.5x_{11}x_{11} + -0.5x_{12}x_{12} + \\
& -0.5x_{13}x_{13} + -0.5x_{14}x_{14} + -0.5x_{15}x_{15} + \\
& -0.5x_{16}x_{16} + -0.5x_{17}x_{17} + -0.5x_{18}x_{18} + \\
& -0.5x_{19}x_{19} + -0.5x_{20}x_{20} + -0.5x_{21}x_{21} \leq 40 \\
& -3x_2 + 7x_3 + -5x_5 + x_6 + x_7 + 2x_9 + -x_{10} + \\
& -x_{11} + -9x_{12} + 3x_{13} + 5x_{14} + x_{17} + 7x_{18} + \\
& -7x_{19} + -4x_{20} + -6x_{21} \leq -5 \\
& 7x_2 + -5x_4 + x_5 + x_6 + 2x_8 + -x_9 + -x_{10} + \\
& -9x_{11} + 3x_{12} + 5x_{13} + x_{16} + 7x_{17} + -7x_{18} + \\
& -4x_{19} + -6x_{20} + -3x_{21} \leq 2 \\
& -5x_3 + x_4 + x_5 + 2x_7 + -x_8 + -x_9 + -9x_{10} + \\
& 3x_{11} + 5x_{12} + x_{15} + 7x_{16} + -7x_{17} + -4x_{18} + \\
& -6x_{19} + -3x_{20} + 7x_{21} \leq -1
\end{aligned}
$$

$-5x_2 + x_3 + x_4 + 2x_6 + -x_7 + -x_8 + -9x_9 +$

$3x_{10} + 5x_{11} + x_{14} + 7x_{15} + -7x_{16} + -4x_{17} +$

$-6x_{18} + -3x_{19} + 7x_{20} \leq -3$

$x_2 + x_3 + 2x_5 + -x_6 + -x_7 + -9x_8 + 3x_9 +$

$5x_{10} + x_{13} + 7x_{14} + -7x_{15} + -4x_{16} + -6x_{17} +$

$-3x_{18} + 7x_{19} + -5x_{21} \leq 5$

$x_2 + 2x_4 + -x_5 + -x_6 + -9x_7 + 3x_8 + 5x_9 +$

$x_{12} + 7x_{13} + -7x_{14} + -4x_{15} + -6x_{16} + -3x_{17} +$

$7x_{18} + -5x_{20} + x_{21} \leq 4$

$2x_3 + -x_4 + -x_5 + -9x_6 + 3x_7 + 5x_8 + x_{11} +$

$7x_{12} + -7x_{13} + -4x_{14} + -6x_{15} + -3x_{16} + 7x_{17} +$

$-5x_{19} + x_{20} + x_{21} \leq -1$

$2x_2 + -x_3 + -x_4 + -9x_5 + 3x_6 + 5x_7 + x_{10} +$

$7x_{11} + -7x_{12} + -4x_{13} + -6x_{14} + -3x_{15} + 7x_{16} +$

$-5x_{18} + x_{19} + x_{20} \leq 0$

$-x_2 + -x_3 + -9x_4 + 3x_5 + 5x_6 + x_9 + 7x_{10} +$

$-7x_{11} + -4x_{12} + -6x_{13} + -3x_{14} + 7x_{15} + -5x_{17} +$

$x_{18} + x_{19} + 2x_{21} \leq 9$

$x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 +$

$x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} +$

$x_{18} + x_{19} + x_{20} + x_{21} \leq 40$

$$-10000 \le x_1 \le 0$$

$$0 \le x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \le 40$$

$$0 \le x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21} \le 40$$

Solution

Objective	-394.7506124	x_1	-394.7506124	x_2	0
x_3	0	x_4	28.80239734	x_5	0
x_6	0	x_7	4.179207216	x_8	0
x_9	0	x_{10}	0	x_{11}	0
x_{12}	0	x_{13}	0	x_{14}	0
x_{15}	0	x_{16}	0.6186802753	x_{17}	4.093282696
x_{18}	0	x_{19}	2.306432471	x_{20}	0
x_{21}	0				

Filename: `fp_2_7_2.def`

Description: Chapter 2, Test Problem 7, Case 2 [22] (with nonrestrictive variable bounds added), a concave quadratic program

$$\min \quad x_1 \tag{275}$$

$$\begin{aligned}
\text{s.t.} \quad & -x_1 + -5x_2 + -5x_3 + -5x_4 + -5x_5 + -5x_6 + \\
& -5x_7 + -5x_8 + -5x_9 + -5x_{10} + -5x_{11} + -5x_{12} + \\
& -5x_{13} + -5x_{14} + -5x_{15} + -5x_{16} + -5x_{17} + -5x_{18} + \\
& -5x_{19} + -5x_{20} + -5x_{21} + -0.5x_2x_2 + -0.5x_3x_3 + \\
& -0.5x_4x_4 + -0.5x_5x_5 + -0.5x_6x_6 + \\
& -0.5x_7x_7 + -0.5x_8x_8 + -0.5x_9x_9 + \\
& -0.5x_{10}x_{10} + -0.5x_{11}x_{11} + -0.5x_{12}x_{12} + \\
& -0.5x_{13}x_{13} + -0.5x_{14}x_{14} + -0.5x_{15}x_{15} + \\
& -0.5x_{16}x_{16} + -0.5x_{17}x_{17} + -0.5x_{18}x_{18} + \\
& -0.5x_{19}x_{19} + -0.5x_{20}x_{20} + -0.5x_{21}x_{21} \le 250
\end{aligned}$$

$-3x_2 + 7x_3 + -5x_5 + x_6 + x_7 + 2x_9 + -x_{10} +$

$-x_{11} + -9x_{12} + 3x_{13} + 5x_{14} + x_{17} + 7x_{18} +$

$-7x_{19} + -4x_{20} + -6x_{21} \leq -5$

$7x_2 + -5x_4 + x_5 + x_6 + 2x_8 + -x_9 + -x_{10} +$

$-9x_{11} + 3x_{12} + 5x_{13} + x_{16} + 7x_{17} + -7x_{18} +$

$-4x_{19} + -6x_{20} + -3x_{21} \leq 2$

$-5x_3 + x_4 + x_5 + 2x_7 + -x_8 + -x_9 + -9x_{10} +$

$3x_{11} + 5x_{12} + x_{15} + 7x_{16} + -7x_{17} + -4x_{18} +$

$-6x_{19} + -3x_{20} + 7x_{21} \leq -1$

$-5x_2 + x_3 + x_4 + 2x_6 + -x_7 + -x_8 + -9x_9 +$

$3x_{10} + 5x_{11} + x_{14} + 7x_{15} + -7x_{16} + -4x_{17} +$

$-6x_{18} + -3x_{19} + 7x_{20} \leq -3$

$x_2 + x_3 + 2x_5 + -x_6 + -x_7 + -9x_8 + 3x_9 +$

$5x_{10} + x_{13} + 7x_{14} + -7x_{15} + -4x_{16} + -6x_{17} +$

$-3x_{18} + 7x_{19} + -5x_{21} \leq 5$

$x_2 + 2x_4 + -x_5 + -x_6 + -9x_7 + 3x_8 + 5x_9 +$

$x_{12} + 7x_{13} + -7x_{14} + -4x_{15} + -6x_{16} + -3x_{17} +$

$7x_{18} + -5x_{20} + x_{21} \leq 4$

$2x_3 + -x_4 + -x_5 + -9x_6 + 3x_7 + 5x_8 + x_{11} +$

$7x_{12} + -7x_{13} + -4x_{14} + -6x_{15} + -3x_{16} + 7x_{17} +$

$-5x_{19} + x_{20} + x_{21} \leq -1$

$$2x_2 + -x_3 + -x_4 + -9x_5 + 3x_6 + 5x_7 + x_{10} +$$

$$7x_{11} + -7x_{12} + -4x_{13} + -6x_{14} + -3x_{15} + 7x_{16} +$$

$$-5x_{18} + x_{19} + x_{20} \leq 0$$

$$-x_2 + -x_3 + -9x_4 + 3x_5 + 5x_6 + x_9 + 7x_{10} +$$

$$-7x_{11} + -4x_{12} + -6x_{13} + -3x_{14} + 7x_{15} + -5x_{17} +$$

$$x_{18} + x_{19} + 2x_{21} \leq 9$$

$$x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 +$$

$$x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} +$$

$$x_{17} + x_{18} + x_{19} + x_{20} + x_{21} \leq 40$$

$$-10000 \leq x_1 \leq 0$$

$$0 \leq x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \leq 40$$

$$0 \leq x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21} \leq 40$$

Solution

Objective	-884.7506124	x_1	-884.7506124	x_2	0
x_3	0	x_4	28.80239734	x_5	0
x_6	0	x_7	4.179207216	x_8	0
x_9	0	x_{10}	0	x_{11}	0
x_{12}	0	x_{13}	0	x_{14}	0
x_{15}	0	x_{16}	0.6186802753	x_{17}	4.093282696
x_{18}	0	x_{19}	2.306432471	x_{20}	0
x_{21}	0				

Filename: `fp_2_7_3.def`

Description: Chapter 2, Test Problem 7, Case 3 [22] (with nonrestrictive variable bounds added), a concave quadratic program

$$\min \quad x_1 \tag{276}$$

$$\text{s.t.} \quad -x_1 + -10x_2x_2 + -10x_3x_3 + -10x_4x_4 +$$

$-10x_5x_5 + -10x_6x_6 + -10x_7x_7 + -10x_8x_8 +$

$-10x_9x_9 + -10x_{10}x_{10} + -10x_{11}x_{11} +$

$-10x_{12}x_{12} + -10x_{13}x_{13} + -10x_{14}x_{14} +$

$-10x_{15}x_{15} + -10x_{16}x_{16} + -10x_{17}x_{17} +$

$-10x_{18}x_{18} + -10x_{19}x_{19} + -10x_{20}x_{20} +$

$-10x_{21}x_{21} \leq 0$

$-3x_2 + 7x_3 + -5x_5 + x_6 + x_7 + 2x_9 + -x_{10} +$

$-x_{11} + -9x_{12} + 3x_{13} + 5x_{14} + x_{17} + 7x_{18} +$

$-7x_{19} + -4x_{20} + -6x_{21} \leq -5$

$7x_2 + -5x_4 + x_5 + x_6 + 2x_8 + -x_9 + -x_{10} +$

$-9x_{11} + 3x_{12} + 5x_{13} + x_{16} + 7x_{17} + -7x_{18} +$

$-4x_{19} + -6x_{20} + -3x_{21} \leq 2$

$-5x_3 + x_4 + x_5 + 2x_7 + -x_8 + -x_9 + -9x_{10} +$

$3x_{11} + 5x_{12} + x_{15} + 7x_{16} + -7x_{17} + -4x_{18} +$

$-6x_{19} + -3x_{20} + 7x_{21} \leq -1$

$-5x_2 + x_3 + x_4 + 2x_6 + -x_7 + -x_8 + -9x_9 +$

$3x_{10} + 5x_{11} + x_{14} + 7x_{15} + -7x_{16} + -4x_{17} +$

$-6x_{18} + -3x_{19} + 7x_{20} \leq -3$

$x_2 + x_3 + 2x_5 + -x_6 + -x_7 + -9x_8 + 3x_9 +$

$5x_{10} + x_{13} + 7x_{14} + -7x_{15} + -4x_{16} + -6x_{17} +$

$-3x_{18} + 7x_{19} + -5x_{21} \leq 5$

$$x_2 + 2x_4 + -x_5 + -x_6 + -9x_7 + 3x_8 + 5x_9 +$$

$$x_{12} + 7x_{13} + -7x_{14} + -4x_{15} + -6x_{16} + -3x_{17} +$$

$$7x_{18} + -5x_{20} + x_{21} \leq 4$$

$$2x_3 + -x_4 + -x_5 + -9x_6 + 3x_7 + 5x_8 + x_{11} +$$

$$7x_{12} + -7x_{13} + -4x_{14} + -6x_{15} + -3x_{16} + 7x_{17} +$$

$$-5x_{19} + x_{20} + x_{21} \leq -1$$

$$2x_2 + -x_3 + -x_4 + -9x_5 + 3x_6 + 5x_7 + x_{10} +$$

$$7x_{11} + -7x_{12} + -4x_{13} + -6x_{14} + -3x_{15} + 7x_{16} +$$

$$-5x_{18} + x_{19} + x_{20} \leq 0$$

$$-x_2 + -x_3 + -9x_4 + 3x_5 + 5x_6 + x_9 + 7x_{10} +$$

$$-7x_{11} + -4x_{12} + -6x_{13} + -3x_{14} + 7x_{15} + -5x_{17} +$$

$$x_{18} + x_{19} + 2x_{21} \leq 9$$

$$x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 +$$

$$x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} +$$

$$x_{18} + x_{19} + x_{20} + x_{21} \leq 40$$

$$-10000 \leq x_1 \leq 0$$

$$0 \leq x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \leq 40$$

$$0 \leq x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21} \leq 40$$

Solution

Objective	-8695.012248	x_1	-8695.012248	x_2	0
x_3	0	x_4	28.80239734	x_5	0
x_6	0	x_7	4.179207216	x_8	0
x_9	0	x_{10}	0	x_{11}	0
x_{12}	0	x_{13}	0	x_{14}	0
x_{15}	0	x_{16}	0.6186802753	x_{17}	4.093282696
x_{18}	0	x_{19}	2.306432471	x_{20}	0
x_{21}	0				

Filename: `fp_2_7_4.def`

Description: Chapter 2, Test Problem 7, Case 4 [22] (with nonrestrictive variable bounds added), a concave quadratic program

$$\begin{aligned}
\min \quad & x_1 && (277)\\
\text{s.t.} \quad & -x_1 + 8x_2 + 8x_3 + 8x_4 + 8x_5 + 8x_6 + 8x_7 + \\
& 8x_8 + 8x_9 + 8x_{10} + 8x_{11} + 8x_{12} + 8x_{13} + \\
& 8x_{14} + 8x_{15} + 8x_{16} + 8x_{17} + 8x_{18} + 8x_{19} + \\
& 8x_{20} + 8x_{21} + -0.5x_2x_2 + -0.5x_3x_3 + \\
& -0.5x_4x_4 + -0.5x_5x_5 + -0.5x_6x_6 + \\
& -0.5x_7x_7 + -0.5x_8x_8 + -0.5x_9x_9 + \\
& -0.5x_{10}x_{10} + -0.5x_{11}x_{11} + -0.5x_{12}x_{12} + \\
& -0.5x_{13}x_{13} + -0.5x_{14}x_{14} + -0.5x_{15}x_{15} + \\
& -0.5x_{16}x_{16} + -0.5x_{17}x_{17} + -0.5x_{18}x_{18} + \\
& -0.5x_{19}x_{19} + -0.5x_{20}x_{20} + -0.5x_{21}x_{21} \leq 640 \\
& -3x_2 + 7x_3 + -5x_5 + x_6 + x_7 + 2x_9 + -x_{10} + \\
& -x_{11} + -9x_{12} + 3x_{13} + 5x_{14} + x_{17} + 7x_{18} + \\
& -7x_{19} + -4x_{20} + -6x_{21} \leq -5
\end{aligned}$$

$7x_2 + -5x_4 + x_5 + x_6 + 2x_8 + -x_9 + -x_{10} +$

$-9x_{11} + 3x_{12} + 5x_{13} + x_{16} + 7x_{17} + -7x_{18} +$

$-4x_{19} + -6x_{20} + -3x_{21} \leq 2$

$-5x_3 + x_4 + x_5 + 2x_7 + -x_8 + -x_9 + -9x_{10} +$

$3x_{11} + 5x_{12} + x_{15} + 7x_{16} + -7x_{17} + -4x_{18} +$

$-6x_{19} + -3x_{20} + 7x_{21} \leq -1$

$-5x_2 + x_3 + x_4 + 2x_6 + -x_7 + -x_8 + -9x_9 +$

$3x_{10} + 5x_{11} + x_{14} + 7x_{15} + -7x_{16} + -4x_{17} +$

$-6x_{18} + -3x_{19} + 7x_{20} \leq -3$

$x_2 + x_3 + 2x_5 + -x_6 + -x_7 + -9x_8 + 3x_9 +$

$5x_{10} + x_{13} + 7x_{14} + -7x_{15} + -4x_{16} + -6x_{17} +$

$-3x_{18} + 7x_{19} + -5x_{21} \leq 5$

$x_2 + 2x_4 + -x_5 + -x_6 + -9x_7 + 3x_8 + 5x_9 +$

$x_{12} + 7x_{13} + -7x_{14} + -4x_{15} + -6x_{16} + -3x_{17} +$

$7x_{18} + -5x_{20} + x_{21} \leq 4$

$2x_3 + -x_4 + -x_5 + -9x_6 + 3x_7 + 5x_8 + x_{11} +$

$7x_{12} + -7x_{13} + -4x_{14} + -6x_{15} + -3x_{16} + 7x_{17} +$

$-5x_{19} + x_{20} + x_{21} \leq -1$

$2x_2 + -x_3 + -x_4 + -9x_5 + 3x_6 + 5x_7 + x_{10} +$

$7x_{11} + -7x_{12} + -4x_{13} + -6x_{14} + -3x_{15} + 7x_{16} +$

$-5x_{18} + x_{19} + x_{20} \leq 0$

$$-x_2 + -x_3 + -9x_4 + 3x_5 + 5x_6 + x_9 + 7x_{10} +$$

$$-7x_{11} + -4x_{12} + -6x_{13} + -3x_{14} + 7x_{15} +$$

$$-5x_{17} + x_{18} + x_{19} + 2x_{21} \leq 9$$

$$x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 +$$

$$x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} +$$

$$x_{18} + x_{19} + x_{20} + x_{21} \leq 40$$

$$-10000 \leq x_1 \leq 0$$

$$0 \leq x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \leq 40$$

$$0 \leq x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21} \leq 40$$

Solution

Objective	-754.7506124	x_1	-754.7506124	x_2	0
x_3	0	x_4	28.80239734	x_5	0
x_6	0	x_7	4.179207216	x_8	0
x_9	0	x_{10}	0	x_{11}	0
x_{12}	0	x_{13}	0	x_{14}	0
x_{15}	0	x_{16}	0.6186802753	x_{17}	4.093282696
x_{18}	0	x_{19}	2.306432471	x_{20}	0
x_{21}	0				

Filename: `fp_2_7_5.def`

Description: Chapter 2, Test Problem 7, Case 5 [22] (with nonrestrictive variable bounds added), a concave quadratic program

$$\min \quad x_1 \tag{278}$$

$$\text{s.t.} \quad -x_1 + 2x_2 + 4x_3 + 6x_4 + 8x_5 + 10x_6 + 12x_7 +$$

$$14x_8 + 16x_9 + 18x_{10} + 20x_{11} + 22x_{12} + 24x_{13} +$$

$$26x_{14} + 28x_{15} + 30x_{16} + 32x_{17} + 34x_{18} + 36x_{19} +$$

$$38x_{20} + 40x_{21} + -0.5x_2x_2 + -x_3x_3 +$$

$$-1.5x_4x_4 + -2x_5x_5 + -2.5x_6x_6 + -3x_7x_7 +$$

$$-3.5x_8x_8 + -4x_9x_9 + -4.5x_{10}x_{10} + -5x_{11}x_{11} +$$

$$-5.5x_{12}x_{12} + -6x_{13}x_{13} + -6.5x_{14}x_{14} +$$

$$-7x_{15}x_{15} + -7.5x_{16}x_{16} + -8x_{17}x_{17} +$$

$$-8.5x_{18}x_{18} + -9x_{19}x_{19} + -9.5x_{20}x_{20} +$$

$$-10x_{21}x_{21} \leq 420$$

$$-3x_2 + 7x_3 + -5x_5 + x_6 + x_7 + 2x_9 + -x_{10} +$$

$$-x_{11} + -9x_{12} + 3x_{13} + 5x_{14} + x_{17} + 7x_{18} +$$

$$-7x_{19} + -4x_{20} + -6x_{21} \leq -5$$

$$7x_2 + -5x_4 + x_5 + x_6 + 2x_8 + -x_9 + -x_{10} +$$

$$-9x_{11} + 3x_{12} + 5x_{13} + x_{16} + 7x_{17} + -7x_{18} +$$

$$-4x_{19} + -6x_{20} + -3x_{21} \leq 2$$

$$-5x_3 + x_4 + x_5 + 2x_7 + -x_8 + -x_9 + -9x_{10} +$$

$$3x_{11} + 5x_{12} + x_{15} + 7x_{16} + -7x_{17} + -4x_{18} +$$

$$-6x_{19} + -3x_{20} + 7x_{21} \leq -1$$

$$-5x_2 + x_3 + x_4 + 2x_6 + -x_7 + -x_8 + -9x_9 +$$

$$3x_{10} + 5x_{11} + x_{14} + 7x_{15} + -7x_{16} + -4x_{17} +$$

$$-6x_{18} + -3x_{19} + 7x_{20} \leq -3$$

$$x_2 + x_3 + 2x_5 + -x_6 + -x_7 + -9x_8 + 3x_9 +$$

$$5x_{10} + x_{13} + 7x_{14} + -7x_{15} + -4x_{16} + -6x_{17} +$$

$$-3x_{18} + 7x_{19} + -5x_{21} \leq 5$$

$$x_2 + 2x_4 + -x_5 + -x_6 + -9x_7 + 3x_8 + 5x_9 +$$

$$x_{12} + 7x_{13} + -7x_{14} + -4x_{15} + -6x_{16} + -3x_{17} +$$

$$7x_{18} + -5x_{20} + x_{21} \leq 4$$

$$2x_3 + -x_4 + -x_5 + -9x_6 + 3x_7 + 5x_8 + x_{11} +$$

$$7x_{12} + -7x_{13} + -4x_{14} + -6x_{15} + -3x_{16} + 7x_{17} +$$

$$-5x_{19} + x_{20} + x_{21} \leq -1$$

$$2x_2 + -x_3 + -x_4 + -9x_5 + 3x_6 + 5x_7 + x_{10} +$$

$$7x_{11} + -7x_{12} + -4x_{13} + -6x_{14} + -3x_{15} + 7x_{16} +$$

$$-5x_{18} + x_{19} + x_{20} \leq 0$$

$$-x_2 + -x_3 + -9x_4 + 3x_5 + 5x_6 + x_9 + 7x_{10} +$$

$$-7x_{11} + -4x_{12} + -6x_{13} + -3x_{14} + 7x_{15} + -5x_{17} +$$

$$x_{18} + x_{19} + 2x_{21} \leq 9$$

$$x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 +$$

$$x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} +$$

$$x_{17} + x_{18} + x_{19} + x_{20} + x_{21} \leq 40$$

$$-10000 \leq x_1 \leq 0$$

$$0 \leq x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \leq 40$$

$$0 \leq x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21} \leq 40$$

The global minimum of this problem is different that the best known point presented in [22]. Table 24 shows the point found by both versions of the algorithm presented in this thesis; this confirms the result reported in [90].

Objective	-4150.4101	x_{11}	0.0000
x_1	-4150.4101	x_{12}	1.7467
x_2	0.0000	x_{13}	0.0000
x_3	0.0000	x_{14}	0.4315
x_4	1.0429	x_{15}	0.0000
x_5	0.0000	x_{16}	0.0000
x_6	0.0000	x_{17}	4.4331
x_7	0.0000	x_{18}	0.0000
x_8	0.0000	x_{19}	15.8589
x_9	0.0000	x_{20}	0.0000
x_{10}	0.0000	x_{21}	16.4869

Table 24: Global minimum of problem fp_2_7_5

Filename: `fp_2_8.def`

Description: Floudas and Pardalos Collection, Chapter 2, Problem 8 (with nonrestrictive variable bounds added), a minimum concave cost transporation program

$$
\begin{aligned}
\min \quad & x_1 \qquad (279) \\
\text{s.t.} \quad & -x_1 + 300x_2 + 270x_3 + 460x_4 + 800x_5 + 740x_6 + 600x_7 + \\
& 540x_8 + 380x_9 + 300x_{10} + 490x_{11} + 380x_{12} + 760x_{13} + \\
& 430x_{14} + 250x_{15} + 390x_{16} + 600x_{17} + 210x_{18} + 830x_{19} + \\
& 470x_{20} + 680x_{21} + 360x_{22} + 290x_{23} + 400x_{24} + 310x_{25} + \\
& -7x_2x_2 + -4x_3x_3 + -6x_4x_4 + -8x_5x_5 + \\
& -12x_6x_6 + -9x_7x_7 + -14x_8x_8 + -7x_9x_9 + -13x_{10}x_{10} + \\
& -12x_{11}x_{11} + -8x_{12}x_{12} + -4x_{13}x_{13} + -7x_{14}x_{14} + \\
& -9x_{15}x_{15} + -16x_{16}x_{16} + -8x_{17}x_{17} + -4x_{18}x_{18} + \\
& -10x_{19}x_{19} + -21x_{20}x_{20} + -13x_{21}x_{21} + -17x_{22}x_{22} +
\end{aligned}
$$

$$-9x_{23}x_{23} + -8x_{24}x_{24} + -4x_{25}x_{25} \leq 0$$

$$x_2 + x_6 + x_{10} + x_{14} + x_{18} + x_{22} = 29$$

$$x_3 + x_7 + x_{11} + x_{15} + x_{19} + x_{23} = 41$$

$$x_4 + x_8 + x_{12} + x_{16} + x_{20} + x_{24} = 13$$

$$x_5 + x_9 + x_{13} + x_{17} + x_{21} + x_{25} = 21$$

$$x_2 + x_3 + x_4 + x_5 = 8$$

$$x_6 + x_7 + x_8 + x_9 = 24$$

$$x_{10} + x_{11} + x_{12} + x_{13} = 20$$

$$x_{14} + x_{15} + x_{16} + x_{17} = 24$$

$$x_{18} + x_{19} + x_{20} + x_{21} = 16$$

$$x_{22} + x_{23} + x_{24} + x_{25} = 12$$

$$-100000 \leq x_1 \leq 100000$$

$$0 \leq x_2, x_3, x_4, x_5 \leq 8$$

$$0 \leq x_6, x_7 \leq 24$$

$$0 \leq x_8, x_{12}, x_{16}, x_{20} \leq 13$$

$$0 \leq x_9, x_{17} \leq 21$$

$$0 \leq x_{10}, x_{11}, x_{13} \leq 20$$

$$0 \leq x_{14}, x_{15} \leq 24$$

$$0 \leq x_{18}, x_{19}, x_{21} \leq 16$$

$$0 \leq x_{22}, x_{23}, x_{24}, x_{25} \leq 12$$

The solution printed in [22] is not feasible. Table 25 reports the solution found by

Objective	15990	x_{13}	0
x_1	15990	x_{14}	0
x_2	3	x_{15}	24
x_3	5	x_{16}	0
x_4	0	x_{17}	0
x_5	0	x_{18}	6
x_6	0	x_{19}	0
x_7	0	x_{20}	10
x_8	3	x_{21}	0
x_9	21	x_{22}	0
x_{10}	20	x_{23}	12
x_{11}	0	x_{24}	0
x_{12}	0	x_{25}	0

Table 25: Global minimum of problem fp_2_8

the algorithm presented.

Filename: `fp_3_1.def`

Description: Chapter 3, Test Problem 1 [22], a quadratically constrainted problem

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 && (280)\\
\text{s.t.} \quad & 0.0025x_4 + 0.0025x_6 \leq 1 \\
& -0.0025x_4 + 0.0025x_5 + 0.0025x_7 \leq 1 \\
& -0.01x_5 + 0.01x_8 \leq 1 \\
& 100x_1 + 833.333x_4 + -x_1x_6 \leq 83333.3 \\
& -1250x_4 + 1250x_5 + x_2x_4 + -x_2x_7 \leq 0 \\
& -2500x_5 + x_3x_5 + -x_3x_8 \leq -1.25 \times 10^6 \\
& 100 \leq x_1 \leq 10000 \\
& 1000 \leq x_2, x_3 \leq 10000 \\
& 10 \leq x_4, x_5, x_6, x_7, x_8 \leq 1000
\end{aligned}
$$

Solution

Objective	7049.248018	x_1	579.2770328	x_2	1359.958638
x_3	5110.012347	x_4	182.0152233	x_5	295.5995061
x_6	217.9847767	x_7	286.4157171	x_8	395.5995061

Filename: `fp_3_2.def`

Description: Chapter 3, Test Problem 2 [22], a quadratic objective function with quadratic constraints

$$
\begin{aligned}
\min \quad & x_1 && (281)\\
\text{s.t.} \quad & -x_1 + 37.2932x_2 + 0.835689x_2x_6 + 5.35785x_4x_4 \leq 40792.1 \\
& -0.0022053x_4x_6 + 0.0056858x_3x_6 + 0.0006262x_2x_5 \leq 6.66559 \\
& 0.0022053x_4x_6 + -0.0056858x_3x_6 + -0.0006262x_2x_5 \leq 85.3344 \\
& 0.0071317x_3x_6 + 0.0021813x_4x_4 + 0.0029955x_2x_3 \leq 29.4875 \\
& -0.0071317x_3x_6 + -0.0021813x_4x_4 + -0.0029955x_2x_3 \leq -9.48751 \\
& 0.0047026x_4x_6 + 0.0019085x_4x_5 + 0.0012547x_2x_4 \leq 15.699 \\
& -0.0047026x_4x_6 + -0.0019085x_4x_5 + -0.0012547x_2x_4 \leq -10.699 \\
& -41000 \leq x_1 \leq -20000 \\
& 78 \leq x_2 \leq 102 \\
& 33 \leq x_3, x_4, x_5, x_6 \leq 45
\end{aligned}
$$

Solution

Objective	-30665.53867	x_1	-30665.53867	x_2	78
x_3	33	x_4	29.99525602	x_5	45
x_6	36.77581291				

Filename: `fp_3_3.def`

Description: Chapter 3, Test Problem 3 [22] (with nonrestrictive variable bounds added), a concave quadratic objective function with quadratic constraints.

$$
\begin{aligned}
\min \quad & x_1 && (282)\\
\text{s.t.} \quad & -x_1 + 100x_2 + 4x_3 + 2x_4 + 8x_5 + 2x_6 + 8x_7 + -25x_2x_2 + \\
& -x_3x_3 + -x_4x_4 + -x_5x_5 + -x_6x_6 + -x_7x_7 \leq 138 \\
& -5 \leq -6x_4 + x_5 + x_4x_4 \\
& -5 \leq -6x_6 + x_7 + x_6x_6 \\
& x_2 + -3x_3 \leq 2 \\
& -x_2 + x_3 \leq 2 \\
& x_2 + x_3 \leq 6 \\
& 2 \leq x_2 + x_3 \\
& -1000 \leq x_1 \leq 0 \\
& 0 \leq x_2, x_3, x_5 \leq 6 \\
& 1 \leq x_4, x_6 \leq 5 \\
& 0 \leq x_7 \leq 10
\end{aligned}
$$

Solution

Objective	-310	x_1	-310	x_2	5
x_3	1	x_4	5	x_5	0
x_6	5	x_7	10		

Filename: `fp_3_4.def`

Description: Chapter 3, Test Problem 4 [22] (with nonrestrictive variable bounds), a linear objective with one reverse convex constraint

$$
\begin{aligned}
\min \quad & -2x_1 + x_2 + -x_3 && (283)\\
\text{s.t.} \quad & x_1 + x_2 + x_3 \leq 4 \\
& 3x_2 + x_3 \leq 6
\end{aligned}
$$

$$-24 \leq -20x_1 + 9x_2 + -13x_3 + 4x_1x_1 + -2x_1x_2 + 2x_1x_3 +$$

$$-2x_2x_1 + 2x_2x_2 + -x_2x_3 + 2x_3x_1 + -x_3x_2 + 2x_3x_3$$

$$0 \leq x_1, x_2 \leq 2$$

$$0 \leq x_3 \leq 3$$

Table 26 shows the alternative global minimum found by the algorithm presented.

Objective	-4
x_1	2
x_2	0
x_3	0

Table 26: Alternative global minimum to problem fp_3_4

Filename: `fp_4_3.def`

Description: Chapter 4, Test Problem 3 [22] (with nonrestrictive variable bounds added)

$$\begin{aligned} \min \quad & x_1 && (284) \\ \text{s.t.} \quad & -x_1 + -6x_2 + -4x_4 + 3x_5 + x_2^{0.6} + x_3^{0.6} \leq 0 \\ & -3x_2 + x_3 + -3x_4 = 0 \\ & x_2 + 2x_4 \leq 4 \\ & x_3 + 2x_5 \leq 4 \\ & -30 \leq x_1 \leq 10 \\ & 0 \leq x_2 \leq 3 \\ & 0 \leq x_3 \leq 4 \end{aligned}$$

$0 \leq x_4 \leq 2$

$0 \leq x_5 \leq 1$

Solution

Objective	-4.514201651	x_1	-4.514201651	x_2	1.333333333
x_3	4	x_4	0	x_5	0

Filename: `fp_4_4.def`

Description: Chapter 4, Test Problem 4 [22] (with nonrestrictive variable bounds added)

$$\begin{aligned} \min \quad & x_1 && (285)\\ \text{s.t.} \quad & -x_1 + -2x_3 + 2x_4 + -x_5 + x_2^{0.6} + 2x_3^{0.6} \leq 0 \\ & -3x_2 + x_3 + -3x_4 = 0 \\ & x_2 + 2x_4 \leq 4 \\ & x_3 + x_5 \leq 4 \\ & -30 \leq x_1 \leq 10 \\ & 0 \leq x_2 \leq 3 \\ & 0 \leq x_3 \leq 4 \\ & 0 \leq x_4, x_5 \leq 2 \end{aligned}$$

Solution

Objective	-2.216804941	x_1	-2.216804941	x_2	1.333333333
x_3	4	x_4	0	x_5	0

Filename: `fp_4_5.def`

Description: Chapter 4, Test Problem 5 [22] (with nonrestrictive variable bounds added)

$$\begin{aligned}
\min \quad & x_1 \\
\text{s.t.} \quad & -x_1 + -4x_4 + 2x_5 + 5x_6 + -x_7 + x_2^{0.6} + x_3^{0.6} + x_4^{0.4} \leq 0 \\
& -3x_2 + x_3 + -3x_5 = 0 \\
& -2x_3 + x_4 + -2x_6 = 0 \\
& 4x_5 + -x_7 = 0 \\
& x_2 + 2x_5 \leq 4 \\
& x_3 + x_6 \leq 4 \\
& x_4 + x_7 \leq 6 \\
& -100 \leq x_1 \leq 100 \\
& 0 \leq x_2 \leq 3 \\
& 0 \leq x_3, x_4 \leq 4 \\
& 0 \leq x_5, x_6 \leq 2 \\
& 0 \leq x_7 \leq 6
\end{aligned} \tag{286}$$

Objective	-13.4019	x_4	4.0000
x_1	-13.4019	x_5	0.5000
x_2	0.1667	x_6	0
x_3	2.0000	x_7	2.0000

Table 27: Global minimum to problem fp_4_5

Filename: `fp_4_6.def`

Description: Chapter 4, Test Problem 6 [22]

$$\min \quad -x_1 + -x_2 \tag{287}$$

$$\text{s.t.}\quad 0 \leq -x_2 + 2x_1^4 + -8x_1^3 + 8x_1^2 + 2$$

$$0 \leq -x_2 + 4x_1^4 - 32x_1^3 + 88x_1^2 + -96x_1^1 + 36$$

$$0 \leq x_1 \leq 3$$

$$0 \leq x_2 \leq 4$$

Solution

Objective	-5.508013272	x_1	2.329520197	x_2	3.178493074

Filename: `fp_4_7.def`

Description: Chapter 4, Test Problem 7 [22]

$$\min\quad x_1 \tag{288}$$

$$\text{s.t.}\quad -x_1 + -12x_2 + -7x_3 + x_3x_3 \leq 0$$

$$-x_3 + -2x_2^4 0x_2^3 + 2 = 0$$

$$-50 \leq x_1 \leq 9$$

$$0 \leq x_2 \leq 2$$

$$0 \leq x_3 \leq 3$$

Solution

Objective	-16.73889318	x_1	-16.73889318	x_2	0.7175357877
x_3	1.46984329				

Filename: `fp_4_9.def`

Description: Chapter 4, Test Problem 9 [22], a heat exchanger network problem

$$\min\quad x_1 \tag{289}$$

$$\text{s.t.}\quad -x_1 + x_2^{0.6} + x_3^{0.6} + x_4^{0.6} \leq 0$$

$$x_5 + -100000x_8 = -1 \times 10^7$$

$$x_6 + 100000x_8 + -100000x_9 = 0$$

$x_7 + 100000x_9 = 5 \times 10^7$

$x_5 + 100000x_{10} = 3.3 \times 10^7$

$x_6 + 100000x_{11} = 4.4 \times 10^7$

$x_7 + 100000x_{12} = 6.6 \times 10^7$

$x_5 + -120x_2x_{13} = 0$

$x_6 + -80x_3x_{14} = 0$

$x_7 + -40x_4x_{15} = 0$

$x_8 + -x_{11} + x_{16} = 0$

$x_9 + -x_{12} + x_{17} = 0$

$-x_{18} + \ln(x_{10} - 100) = 0$

$-x_{19} + \ln(-x_8 + 300) = 0$

$-x_{20} + \ln(x_{16}) = 0$

$-x_{21} + \ln(-x_9 + 400) = 0$

$-x_{22} + \ln(x_{17}) = 0$

$-x_8 + -x_{10} + x_{13}x_{18} + -x_{13}x_{19} = -400$

$x_8 + -x_9 + -x_{11} + x_{14}x_{20} + -x_{14}x_{21} = -400$

$x_9 + -x_{12} + -4.60517x_{15} + x_{15}x_{22} = -100$

$0 \leq x_1 \leq 20000$

$0 \leq x_2, x_3, x_4 \leq 1 \times 10^6$

$0 \leq x_5, x_6, x_7 \leq 4 \times 10^7$

$100 \leq x_8 \leq 299.99$

$$100 \le x_9 \le 399.99$$

$$100.01 \le x_{10} \le 300$$

$$100 \le x_{11} \le 400$$

$$100 \le x_{12} \le 600$$

$$0 \le x_{13}, x_{14}, x_{15} \le 500$$

$$0.01 \le x_{16} \le 300$$

$$0.01 \le x_{17} \le 400$$

$$-4.7 \le x_{18}, x_{19}, x_{20}, x_{21}, x_{22} \le 6.25$$

Potential solution:

Solution

Objective	12812.5	x_1	12812.5	x_2	722.1602494
x_3	8628.371755	x_4	2193.749851	x_5	9951396.436
x_6	18846563.16	x_7	11202040.4	x_8	199.5139644
x_9	387.979596	x_{10}	230.4860356	x_{11}	251.5343684
x_{12}	547.979596	x_{13}	114.8336587	x_{14}	27.30318607
x_{15}	127.6585887	x_{16}	52.020404	x_{17}	160
x_{18}	4.871266214	x_{19}	4.610018769	x_{20}	3.951636026
x_{21}	2.486605539	x_{22}	5.075173815		

Filename: `fp_5_1.def`

Description: Chapter 5, Test Problem 1 [22], a nonsharp separation of propane, isobutane, and n-butane.

$$\min \quad x_1 \tag{290}$$

$$\begin{aligned}\text{s.t.} \quad & -x_1 + -0.0139904x_6 + -0.0661588x_{14} + 0.0093514x_6x_{46} + \\ & 0.0077308x_6x_{48} + -0.0005719x_6x_{28} + 0.0042656x_6x_{29} + \\ & 0.0338147x_{14}x_{47} + 0.0373349x_{14}x_{49} + \\ & 0.0016371x_{14}x_{37} + 0.0288996x_{14}x_{38} \le -0.99782\end{aligned}$$

$$x_2 + x_3 + x_4 + x_5 = 300$$

$$x_7 + -x_8 + -x_9 = 0$$

$$x_{10} + -x_{11} + -x_{12} + -x_{13} = 0$$

$$x_{15} + -x_{16} + -x_{17} + -x_{18} = 0$$

$$x_{19} + -x_{20} + -x_{21} = 0$$

$$x_7 x_{31} + -x_{46} x_{22} = 0$$

$$x_{15} x_{41} + -x_{47} x_{26} = 0$$

$$x_{10} x_{35} + -x_{48} x_{23} = 0$$

$$x_{19} x_{45} + -x_{49} x_{27} = 0$$

$$x_{22} + -x_6 x_{28} = 0$$

$$x_{23} + -x_6 x_{29} = 0$$

$$x_{24} + -x_6 x_{30} = 0$$

$$x_{25} + -x_{14} x_{37} = 0$$

$$x_{26} + -x_{14} x_{38} = 0$$

$$x_{27} + -x_{14} x_{39} = 0$$

$$x_{22} + -x_7 x_{31} + -x_{10} x_{34} = 0$$

$$x_{23} + -x_7 x_{32} + -x_{10} x_{35} = 0$$

$$x_{24} + -x_7 x_{33} + -x_{10} x_{36} = 0$$

$$x_{25} + -x_{15} x_{40} + -x_{19} x_{43} = 0$$

$$x_{26} + -x_{15} x_{41} + -x_{19} x_{44} = 0$$

$$x_{27} + -x_{15} x_{42} + -x_{19} x_{45} = 0$$

$$0.333333x_2 + -x_{22} + x_{16}x_{40} = 0$$

$$0.333333x_2 + -x_{23} + x_{16}x_{41} = 0$$

$$0.333333x_2 + -x_{24} + x_{16}x_{42} = 0$$

$$0.333333x_3 + -x_{25} + x_{11}x_{34} = 0$$

$$0.333333x_3 + -x_{26} + x_{11}x_{35} = 0$$

$$0.333333x_3 + -x_{27} + x_{11}x_{36} = 0$$

$$x_{33} = 0$$

$$x_{43} = 0$$

$$0.333333x_4 + x_8x_{31} + x_{12}x_{34} + x_{17}x_{40} + x_{20}x_{43} = 30$$

$$0.333333x_4 + x_8x_{32} + x_{12}x_{35} + x_{17}x_{41} + x_{20}x_{44} = 50$$

$$0.333333x_4 + x_8x_{33} + x_{12}x_{36} + x_{17}x_{42} + x_{20}x_{45} = 30$$

$$x_{28} + x_{29} + x_{30} = 1$$

$$x_{31} + x_{32} + x_{33} = 1$$

$$x_{34} + x_{35} + x_{36} = 1$$

$$x_{37} + x_{38} + x_{39} = 1$$

$$x_{40} + x_{41} + x_{42} = 1$$

$$x_{43} + x_{44} + x_{45} = 1$$

$$-100 \leq x_1 \leq 100$$

$$0 \leq x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \leq 300$$

$$0 \leq x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20} \leq 300$$

$$0 \leq x_{21}, x_{22}, x_{23}, x_{24}, x_{25}, x_{26}, x_{27} \leq 300$$

$$0 \leq x_{28}, x_{29}, x_{30}, x_{31}, x_{32}, x_{33}, x_{34}, x_{35} \leq 1$$

$$0 \leq x_{36}, x_{37}, x_{38}, x_{39}, x_{40}, x_{41}, x_{42}, x_{43} \leq 1$$

$$0 \leq x_{44}, x_{45} \leq 1$$

$$0.85 \leq x_{46}, x_{47}, x_{48}, x_{49} \leq 1$$

Filename: `f_g_I.def`

Description: Chapter 6, Test Problem 1, Case I [22], a pool blending problem

$$\begin{aligned}
\min \quad & 6x_1 + 16x_2 + 10x_3 + 10x_4 + -9x_7 + -15x_8 \qquad (291)\\
\text{s.t.} \quad & -x_1 + -x_2 + x_5 + x_6 = 0\\
& -x_3 + -x_5 + x_7 = 0\\
& -x_4 + -x_6 + x_8 = 0\\
& -3x_1 + -x_2 + x_9x_5 + x_9x_6 = 0\\
& 2x_3 + -2.5x_7 + x_9x_5 \leq 0\\
& 2x_4 + -1.5x_8 + x_9x_6 \leq 0\\
& 0 \leq x_1, x_2 \leq 300\\
& 0 \leq x_3, x_5, x_7 \leq 100\\
& 0 \leq x_4, x_6, x_8 \leq 200\\
& 1 \leq x_9 \leq 3
\end{aligned}$$

Solution

Objective	-400	x_1	0	x_2	100
x_3	0	x_4	100	x_5	0
x_6	100	x_7	0	x_8	200
x_9	1				

Filename: `f_g_II.def`

Description: Chapter 6, Test Problem 2 [22], a pool blending problem

$$
\begin{aligned}
\min \quad & 6x_1 + 16x_2 + 10x_3 + 10x_4 + -9x_7 + -15x_8 \\
\text{s.t.} \quad & -x_1 + -x_2 + x_5 + x_6 = 0 \\
& -x_3 + -x_5 + x_7 = 0 \\
& -x_4 + -x_6 + x_8 = 0 \\
& -3x_1 + -x_2 + x_9x_5 + x_9x_6 = 0 \\
& 2x_3 + -2.5x_7 + x_9x_5 \leq 0 \\
& 2x_4 + -1.5x_8 + x_9x_6 \leq 0 \\
& 0 \leq x_1, x_2 \leq 800 \\
& 0 \leq x_3, x_5, x_7 \leq 600 \\
& 0 \leq x_4, x_6, x_8 \leq 200 \\
& 1 \leq x_9 \leq 3
\end{aligned}
\tag{292}
$$

Solution

Objective	-600	x_1	300	x_2	0
x_3	300	x_4	0	x_5	300
x_6	0	x_7	600	x_8	0
x_9	3				

Filename: `f_g_III.def`

Description: Chapter 6, Problem 3 [22], a pool blending problem

$$
\begin{aligned}
\min \quad & 6x_1 + 13x_2 + 10x_3 + 10x_4 + -9x_7 + -15x_8 \\
\text{s.t.} \quad & -x_1 + -x_2 + x_5 + x_6 = 0 \\
& -x_3 + -x_5 + x_7 = 0
\end{aligned}
\tag{293}
$$

$$-x_4 + -x_6 + x_8 = 0$$

$$-3x_1 + -x_2 + x_9 x_5 + x_9 x_6 = 0$$

$$2x_3 + -2.5x_7 + x_9 x_5 \leq 0$$

$$2x_4 + -1.5x_8 + x_9 x_6 \leq 0$$

$$0 \leq x_1, x_2 \leq 300$$

$$0 \leq x_3, x_5, x_7 \leq 100$$

$$0 \leq x_4, x_6, x_8 \leq 200$$

$$1 \leq x_9 \leq 3$$

Solution

Objective	-750	x_1	50	x_2	150
x_3	0	x_4	0	x_5	0
x_6	200	x_7	0	x_8	200
x_9	1.5				

Filename: `diet.def`

Description: Minos Diet Problem [74], a simple LP

$$\min \quad 3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6 \tag{294}$$

$$\text{s.t.} \quad 2000 \leq 110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6$$

$$55 \leq 4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6$$

$$800 \leq 2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6$$

$$0 \leq x_1 \leq 4$$

$$0 \leq x_2 \leq 3$$

$$0 \leq x_3, x_5, x_6 \leq 2$$

$$0 \leq x_4 \leq 8$$

Solution					
Objective	92.5	x_1	4	x_2	0
x_3	0	x_4	4.5	x_5	2
x_6	0				

Filename: `e_1.def`

Description: Minimization inside of a sphere.

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & x_1x_1 + x_2x_2 + x_3x_3 \leq 9 \\ & -4 \leq x_1, x_2, x_3 \leq 4 \end{aligned} \tag{295}$$

Solution					
Objective	-3	x_1	-3	x_2	0
x_3	0				

Filename: `f_a.def`

Description: A problem with one bilinear term in its objective function [24]

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + -x_2 + -x_3 + x_2x_3 \leq 0 \\ & -6x_2 + 8x_3 \leq 3 \\ & 3x_2 + -x_3 \leq 3 \\ & -6 \leq x_1 \leq 9 \\ & 0 \leq x_2, x_3 \leq 3 \end{aligned} \tag{296}$$

Solution					
Objective	-1.083333333	x_1	-1.083333333	x_2	1.166666667
x_3	0.5				

Filename: `f_b.def`

Description: A problem with a bilinear term in a constraint [24]

$$\begin{aligned} \min \quad & -x_1 + -x_2 \\ \text{s.t.} \quad & x_1 x_2 \leq 4 \\ & 0 \leq x_1 \leq 4 \\ & 0 \leq x_2 \leq 8 \end{aligned} \tag{297}$$

Solution

Objective	-8.5	x_1	0.5	x_2	8

Filename: `f_c.def`

Description: Example one from [124] originally from Kono,

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + x_2 + -x_3 + -x_4 + -x_2x_4 + x_2x_5 + x_3x_4 + -x_3x_5 \leq 0 \\ & x_2 + 4x_3 \leq 8 \\ & 4x_2 + x_3 \leq 12 \\ & 3x_2 + 4x_3 \leq 12 \\ & 2x_4 + x_5 \leq 8 \\ & x_4 + 2x_5 \leq 8 \\ & x_4 + x_5 \leq 5 \\ & -10000 \leq x_1 \leq 10000 \\ & 0 \leq x_2 \leq 3 \\ & 0 \leq x_3 \leq 2 \\ & 0 \leq x_4, x_5 \leq 4 \end{aligned} \tag{298}$$

Solution

Objective	-13	x_1	-13	x_2	3
x_3	0	x_4	4	x_5	0

Filename: `f_d.def`

Description: Example two from [124], an indefinite quadratic program

$$
\begin{aligned}
\min \quad & x_1 && (299)\\
\text{s.t.} \quad & -x_1 + -1197x_2 + -405x_3 + -1012x_4 + -4823x_5 + -1890x_6 + \\
& 1300x_7 + -2937x_8 + -1334x_9 + 3526x_{10} + 1558x_{11} + 2184x_{12} + \\
& 294x_{13} + -3888x_{14} + -2730x_{15} + 935x_{16} + -4284x_{17} + \\
& -1647x_{18} + 4941x_{19} + -3686x_{20} + 1898x_{21} + -31.5x_2x_2 + \\
& -7.5x_3x_3 + -22x_4x_4 + -45.5x_5x_5 + -22.5x_6x_6 + \\
& -25x_7x_7 + -44.5x_8x_8 + -29x_9x_9 + -43x_{10}x_{10} + \\
& -41x_{11}x_{11} + 21x_{12}x_{12} + 49x_{13}x_{13} + 24x_{14}x_{14} + \\
& 45.5x_{15}x_{15} + 5.5x_{16}x_{16} + 31.5x_{17}x_{17} + 30.5x_{18}x_{18} + \\
& 30.5x_{19}x_{19} + 19x_{20}x_{20} + 13x_{21}x_{21} \leq -6598.5 \\
& 3x_2 + 5x_3 + 5x_4 + 6x_5 + 4x_6 + 4x_7 + 5x_8 + 6x_9 + \\
& 4x_{10} + 4x_{11} + 8x_{12} + 2x_{13} + 4x_{14} + x_{15} + x_{16} + x_{17} + \\
& 2x_{18} + x_{19} + 7x_{20} + 3x_{21} \leq 380 \\
& 5x_2 + 4x_3 + 5x_4 + 4x_5 + x_6 + 4x_7 + 4x_8 + 2x_9 + \\
& 5x_{10} + 2x_{11} + 3x_{12} + 6x_{13} + x_{14} + 7x_{15} + 7x_{16} + \\
& 5x_{17} + 8x_{18} + 7x_{19} + 2x_{20} + x_{21} \leq 415 \\
& x_2 + 5x_3 + 2x_4 + 4x_5 + 7x_6 + 3x_7 + x_8 + 5x_9 + \\
& 7x_{10} + 6x_{11} + x_{12} + 7x_{13} + 2x_{14} + 4x_{15} + 7x_{16} + \\
& 5x_{17} + 3x_{18} + 4x_{19} + x_{20} + 2x_{21} \leq 385
\end{aligned}
$$

$3x_2 + 2x_3 + 6x_4 + 3x_5 + 2x_6 + x_7 + 6x_8 +$

$x_9 + 7x_{10} + 3x_{11} + 7x_{12} + 7x_{13} + 8x_{14} + 2x_{15} +$

$3x_{16} + 4x_{17} + 5x_{18} + 8x_{19} + x_{20} + 2x_{21} \leq 405$

$6x_2 + 6x_3 + 6x_4 + 4x_5 + 5x_6 + 2x_7 + 2x_8 +$

$4x_9 + 3x_{10} + 2x_{11} + 7x_{12} + 5x_{13} + 3x_{14} + 6x_{15} +$

$7x_{16} + 5x_{17} + 8x_{18} + 4x_{19} + 6x_{20} + 3x_{21} \leq 470$

$5x_2 + 5x_3 + 2x_4 + x_5 + 3x_6 + 5x_7 + 5x_8 +$

$7x_9 + 4x_{10} + 3x_{11} + 4x_{12} + x_{13} + 7x_{14} + 3x_{15} +$

$8x_{16} + 3x_{17} + x_{18} + 6x_{19} + 2x_{20} + 8x_{21} \leq 415$

$3x_2 + 6x_3 + 6x_4 + 3x_5 + x_6 + 6x_7 + x_8 +$

$6x_9 + 7x_{10} + x_{11} + 4x_{12} + 3x_{13} + x_{14} + 4x_{15} +$

$3x_{16} + 6x_{17} + 4x_{18} + 6x_{19} + 5x_{20} + 4x_{21} \leq 400$

$x_2 + 2x_3 + x_4 + 7x_5 + 8x_6 + 7x_7 + 6x_8 +$

$5x_9 + 8x_{10} + 7x_{11} + 2x_{12} + 3x_{13} + 5x_{14} +$

$5x_{15} + 4x_{16} + 5x_{17} + 4x_{18} + 2x_{19} + 2x_{20} + 8x_{21} \leq 460$

$8x_2 + 5x_3 + 2x_4 + 5x_5 + 3x_6 + 8x_7 + x_8 +$

$3x_9 + 3x_{10} + 5x_{11} + 4x_{12} + 5x_{13} + 5x_{14} + 6x_{15} +$

$x_{16} + 7x_{17} + x_{18} + 2x_{19} + 2x_{20} + 4x_{21} \leq 400$

$x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} +$

$x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} +$

$x_{19} + x_{20} + x_{21} \leq 200$

$$-2.03877 \times 10^6 \leq x_1 \leq 1.75199 \times 10^6$$

$$0 \leq x_2, x_7 \leq 50$$

$$0 \leq x_3, x_4 \leq 66.6667$$

$$0 \leq x_5 \leq 63.3333$$

$$0 \leq x_6, x_{10}, x_{13} \leq 55$$

$$0 \leq x_8 \leq 67.5$$

$$0 \leq x_9, x_{15} \leq 59.2857$$

$$0 \leq x_{11} \leq 64.1667$$

$$0 \leq x_{12} \leq 47.5$$

$$0 \leq x_{14} \leq 50.625$$

$$0 \leq x_{16}, x_{18}, x_{21} \leq 51.875$$

$$0 \leq x_{17} \leq 57.1429$$

$$0 \leq x_{19} \leq 50.625$$

$$0 \leq x_{20} \leq 54.2857$$

Solution

Objective	-491746.982	x_1	-491746.982	x_2	0
x_3	0	x_4	0	x_5	62.60869565
x_6	0	x_7	0	x_8	0
x_9	0	x_{10}	0	x_{11}	0
x_{12}	0	x_{13}	0	x_{14}	0
x_{15}	0	x_{16}	0	x_{17}	4.347826087
x_{18}	0	x_{19}	0	x_{20}	0
x_{21}	0				

Filename: `f_e.def`

Description: Hesse's function, a concave objective with linear and quadratic constraints

$$\begin{aligned}
\min \quad & x_1 && (300)\\
\text{s.t.} \quad & -x_1 + 100x_2 + 4x_3 + 2x_4 + 8x_5 + 2x_6 + 8x_7 + -25x_2x_2 + \\
& -x_3x_3 + -x_4x_4 + -x_5x_5 + -x_6x_6 + -x_7x_7 \leq 138 \\
& 2 \leq x_2 + x_3 \\
& x_2 + x_3 \leq 6 \\
& -x_2 + x_3 \leq 2 \\
& x_2 + -3x_3 \leq 2 \\
& -5 \leq -6x_4 + x_5 + x_4x_4 \\
& -5 \leq -6x_6 + x_7 + x_6x_6 \\
& -1 \times 10^6 \leq x_1 \leq 1 \times 10^6 \\
& 0 \leq x_2, x_3, x_5 \leq 6 \\
& 1 \leq x_4, x_6 \leq 5 \\
& 0 \leq x_7 \leq 10
\end{aligned}$$

Solution

Objective	-310	x_1	-310	x_2	5
x_3	1	x_4	5	x_5	0
x_6	5	x_7	10		

Filename: `f_f.def`

Description: Example 4 from [124], a linear objective with two quadratic constraints, one convex and one nonconvex, and two linear constraints

$$\begin{aligned}
\min \quad & x_1 + x_2 && (301)\\
\text{s.t.} \quad & x_1x_1 + x_2x_2 \leq 4 \\
& 1 \leq x_1x_1 + x_2x_2
\end{aligned}$$

$$x_1 + -x_2 \leq 1$$

$$-x_1 + x_2 \leq 1$$

$$-2 \leq x_1, x_2 \leq 2$$

Solution

Objective	-2.828427125	x_1	-1.414215754	x_2	-1.414211371

Filename: `f_wingo.def`

Description: Example 7 from [124] original from Wingo, minimization of a sixth order polynomial

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + 0.166667x_2^6 + -2.08x_2^5 + 0.4875x_2^4 + \\ & 7.1x_2^3 + -3.95x_2^2 + -x_2^1 + 0.1 \leq 0 \\ & -9 \times 10^7 \leq x_1 \leq 9 \times 10^7 \\ & -2 \leq x_2 \leq 11 \end{aligned} \tag{302}$$

Solution

Objective	-29763.23333	x_1	-29763.23333	x_2	10

Filename: `s_1.def`

Description: Example 1 from [114], a linear objective function and two bilinear constraints

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + 2x_2 + x_3 \leq 0 \\ & -16x_2x_3 \leq -1 \\ & -4x_2x_2 + -4x_3x_3 \leq -1 \end{aligned} \tag{303}$$

$$0 \leq x_1 \leq 3$$

$$0 \leq x_2, x_3 \leq 1$$

Solution

Objective	0.7417819582	x_1	0.7417819582	x_2	0.1294095226
x_3	0.4829629131				

Filename: `s_1b.def`

Description: Example 1 from [114] with modified variable bounds

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + 2x_2 + x_3 \leq 0 \\ & -16x_2x_3 \leq -1 \\ & -4x_2x_2 + -4x_3x_3 \leq -1 \\ & 0 \leq x_1 \leq 3 \\ & 0 \leq x_2, x_3 \leq 2 \end{aligned} \tag{304}$$

Solution

Objective	0.7417819582	x_1	0.7417819582	x_2	0.1294095226
x_3	0.4829629131				

Filename: `s_1c.def`

Description: Example 1 from [114] with modified variable bounds

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + 2x_2 + x_3 \leq 0 \\ & -16x_2x_3 \leq -1 \\ & -4x_2x_2 + -4x_3x_3 \leq -1 \\ & 0 \leq x_1 \leq 3 \end{aligned} \tag{305}$$

$$0.12 \leq x_2 \leq 0.14$$

$$0.45 \leq x_3 \leq 0.5$$

Solution

Objective	0.7417819582	x_1	0.7417819582	x_2	0.1294095226
x_3	0.4829629131				

Filename: `s_1d.def`

Description: Example 1 from [114] with modified variable bounds

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + 2x_2 + x_3 \leq 0 \\ & -16x_2x_3 \leq -1 \\ & -4x_2x_2 + -4x_3x_3 \leq -1 \\ & 0 \leq x_1 \leq 3 \\ & 0 \leq x_2 \leq 0.6 \\ & 0.4 \leq x_3 \leq 0.7 \end{aligned} \tag{306}$$

Solution

Objective	0.7417819582	x_1	0.7417819582	x_2	0.1294095225
x_3	0.4829629133				

Filename: `s_2.def`

Description: Example 2 from [114], a bilinear objective with on bilinear constraint

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + -2x_2x_3 \leq 0 \\ & 2x_2 + 2x_3 + 4x_2x_3 \leq 3 \\ & -3 \leq x_1 \leq 10 \\ & 0 \leq x_2, x_3 \leq 1 \end{aligned} \tag{307}$$

Solution

Objective	-0.5	x_1	-0.5	x_2	0.5
x_3	0.5				

Filename: `s_2b.def`

Description: Example 2 from [114] with modified variable bounds

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + -2x_2x_3 \leq 0 \\ & 2x_2 + 2x_3 + 4x_2x_3 \leq 3 \\ & -3 \leq x_1 \leq 10 \\ & 0 \leq x_2, x_3 \leq 1.5 \end{aligned} \tag{308}$$

Solution

Objective	-0.5	x_1	-0.5	x_2	0.5
x_3	0.5				

Filename: `s_2c.def`

Description: Example 2 from [114] with modified variable bounds

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + -2x_2x_3 \leq 0 \\ & 2x_2 + 2x_3 + 4x_2x_3 \leq 3 \\ & -3 \leq x_1 \leq 10 \\ & 0.48 \leq x_2, x_3 \leq 0.52 \end{aligned} \tag{309}$$

Solution

Objective	-0.5000000008	x_1	-0.5000000008	x_2	0.4999999996
x_3	0.4999999996				

Filename: `s_2d.def`

Description: Example 2 from [114] with modified variable bounds

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + -2x_2x_3 \leq 0 \\ & 2x_2 + 2x_3 + 4x_2x_3 \leq 3 \\ & -3 \leq x_1 \leq 10 \\ & 0.25 \leq x_2, x_3 \leq 0.75 \end{aligned} \tag{310}$$

Solution

Objective	-0.5	x_1	-0.5	x_2	0.5
x_3	0.5				

Filename: `s_3.def`

Description: Example 3 from [114], a quadratic objective with a 4'th order polynomial equality constraint

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & -x_1 + -12x_2 + -7x_3 + x_3x_3 \leq 0 \\ & -x_3 + -2x_2^4 = -2 \\ & -40 \leq x_1 \leq 40 \\ & 0 \leq x_2 \leq 2 \\ & 0 \leq x_3 \leq 3 \end{aligned} \tag{311}$$

Solution

Objective	-16.73889319	x_1	-16.73889319	x_2	0.7175370521
x_3	1.469839553				

Filename: `s_4.def`

Description: Example 4 from [114], a heat exchanger network problem original by Westerberg and Shah (1978)

$$\begin{aligned}
\min \quad & x_1 \\
\text{s.t.} \quad & -x_1 + 35x_2^{0.6} + 35x_3^{0.6} \leq 0 \\
& 600x_2 + -50x_4 + -x_2x_4 = -5000 \\
& 600x_3 + 50x_4 = 15000 \\
& 0 \leq x_1 \leq 340 \\
& 0 \leq x_2 \leq 34 \\
& 0 \leq x_3 \leq 17 \\
& 100 \leq x_4 \leq 300
\end{aligned} \tag{312}$$

Solution

Objective	189.3116297	x_1	189.3116297	x_2	0
x_3	16.66666667	x_4	100		

Filename: `s_5.def`

Description: Example 5 from [114]

$$\begin{aligned}
\min \quad & x_1 \\
\text{s.t.} \quad & -x_1 + -6x_2 + -4x_4 + 3x_5 + x_2^{0.6} + x_3^{0.6} \leq 0 \\
& -3x_2 + x_3 + -3x_4 = 0 \\
& x_2 + 2x_4 \leq 4 \\
& x_3 + 2x_5 \leq 4 \\
& -100 \leq x_1 \leq 100 \\
& 0 \leq x_2 \leq 3 \\
& 0 \leq x_3 \leq 4 \\
& 0 \leq x_4 \leq 2 \\
& 0 \leq x_5 \leq 1
\end{aligned} \tag{313}$$

Solution

Objective	-4.514201651	x_1	-4.514201651	x_2	1.333333333
x_3	4	x_4	0	x_5	0

Filename: `s_6.def`

Description: Example 6 from [114], a pool blending problem

$$
\begin{aligned}
\min\quad & x_1 && (314)\\
\text{s.t.}\quad & -x_1 + 6x_2 + 16x_3 + 10x_5 + -9x_6 + 10x_8 + -15x_9 \leq 0\\
& x_2 + x_3 + -x_4 + -x_7 = 0\\
& 0.03x_2 + 0.01x_3 + -x_4x_{10} + -x_7x_{10} = 0\\
& x_4 + x_5 + -x_6 = 0\\
& x_7 + x_8 + -x_9 = 0\\
& 0.02x_5 + -0.025x_6 + x_4x_{10} \leq 0\\
& 0.02x_8 + -0.015x_9 + x_7x_{10} \leq 0\\
& -4000 \leq x_1 \leq 10000\\
& 0 \leq x_2, x_3 \leq 300\\
& 0 \leq x_4, x_5, x_6 \leq 100\\
& 0 \leq x_7, x_8, x_9 \leq 200\\
& 0.01 \leq x_{10} \leq 0.03
\end{aligned}
$$

Solution

Objective	-400	x_1	-400	x_2	0
x_3	100	x_4	0	x_5	0
x_6	0	x_7	100	x_8	100
x_9	200	x_{10}	0.01		

Filename: `w.def`

Description: A heat exchanger problem from [130] using logarithms

$$\begin{aligned}
\min \quad & x_1 \\
\text{s.t.} \quad & -x_1 + 35x_2^{0.6} + 35x_3^{0.6} \leq 0 \\
& -300x_3 + 7500x_5 + -7500x_6 + -25x_4x_5 + 25x_4x_6 + x_3x_4 = 0 \\
& 100x_2 + 155.365x_4 + 2500x_7 + -x_2x_4 + -25x_4x_7 = 15536.5 \\
& -x_5 + \ln(-x_4 + 900) = 0 \\
& -x_6 + \ln(x_4 + 300) = 0 \\
& -x_7 + \ln(-2x_4 + 700) = 0 \\
& 0 \leq x_1 \leq 1000 \\
& 0 \leq x_2, x_3 \leq 40 \\
& 100 \leq x_4 \leq 300 \\
& 6.3 \leq x_5 \leq 6.7 \\
& 5.9 \leq x_6 \leq 6.4 \\
& 4.5 \leq x_7 \leq 6.25
\end{aligned} \tag{315}$$

Solution

Objective	193.7783493	x_1	193.7783493	x_2	0
x_3	17.3272116	x_4	100.0156586	x_5	6.684592154
x_6	5.991503693	x_7	6.214545462		

Filename: `w_b.def`

Description: A second formulation of the heat exchanger network problem from [130]

$$\begin{aligned}
\min \quad & x_1 \\
\text{s.t.} \quad & -x_1 + 35x_2^{0.6} + 35x_3^{0.6} \leq 0 \\
& -300x_3 + 7500x_5 + -7500x_6 + x_3x_4 +
\end{aligned} \tag{316}$$

$$-25x_4 \ln(-x_4 + 900) + 25x_4 \ln(x_4 + 300) = 0$$

$$100x_2 + 155.365x_4 + 2500x_7 + -x_2x_4 +$$

$$-25x_4 \ln(-2x_4 + 700) = 15536.5$$

$$-x_5 + \ln(-x_4 + 900) = 0$$

$$-x_6 + \ln(x_4 + 300) = 0$$

$$-x_7 + \ln(-2x_4 + 700) = 0$$

$$0 \le x_1 \le 1000$$

$$0 \le x_2, x_3 \le 100$$

$$100 \le x_4 \le 300$$

$$6.3 \le x_5 \le 6.7$$

$$5.9 \le x_6 \le 6.4$$

$$4.5 \le x_7 \le 6.25$$

Solution

Objective	193.7828399	x_1	193.7828399	x_2	0
x_3	17.32788086	x_4	100.0085187	x_5	6.684601079
x_6	5.991485844	x_7	6.214574023		

Filename: `w_c.def`

Description: A third formulation of the heat exchanger network problem from [130]

$$\min \quad x_1 \tag{317}$$

$$\text{s.t.} \quad -x_1 + 35x_2^{0.6} + 35x_3^{0.6} \le 0$$

$$-300x_3 + x_3x_4 + -25x_4 \ln(-x_4 + 900) +$$

$$25x_4 \ln(x_4 + 300) + 7500 \ln(-x_4 + 900) + -7500 \ln(x_4 + 300) = 0$$

$100x_2 + 155.365x_4 + -x_2x_4 + -25x_4 \ln(-2x_4 + 700) +$

$2500 \ln(-2x_4 + 700) = 15536.5$

$0 \leq x_1 \leq 1000$

$0 \leq x_2, x_3 \leq 100$

$100 \leq x_4 \leq 300$

Solution

Objective	193.7855896	x_1	193.7855896	x_2	0
x_3	17.32829068	x_4	100.0041475		