1 COMPUTATIONAL EXPERIENCE OF AN INTERIOR-POINT SQP ALGORITHM IN A PARALLEL BRANCH-AND-BOUND FRAMEWORK

Eva K. Lee

School of Industrial and Systems Engineering,
Georgia Institute of Technology,
Atlanta, GA 30332-0205.*

evakylee@isye.gatech.edu

John E. Mitchell

Department of Mathematical Sciences,
Rensselaer Polytechnic Institute,
Troy, New York 12180–3590.†

mitchj@rpi.edu

Abstract: An interior-point algorithm within a parallel branch-and-bound framework for solving nonlinear mixed integer programs is described. The nonlinear programming relaxations at each node are solved using an interior point SQP method. In contrast to solving the relaxation to optimality at each tree node, the relaxation is only solved to near-optimality. Analogous to employing advanced bases in simplex-based linear MIP solvers, a "dynamic" collection of warmstart vectors is kept to provide "advanced warmstarts" at each branch-and-bound node. The code has the capability to run in both shared-memory and distributed-memory parallel environments. Preliminary computational results on various classes of linear mixed integer programs and quadratic portfolio problems are presented.

*Supported in part by NSF/NATO grant GER-9452935, NSF grant CCR-9501584, and SUN AEG EDUD-US-970311.
†Supported in part by ONR grant N00014–94–1–0391, and by a grant from the Dutch NWO and by Delft University of Technology for 1997–98, while visiting TWI/SSOR at Delft University of Technology.

1.1 INTRODUCTION

Branch-and-bound is a classical approach for solving *linear* mixed integer programs. While the approach is applicable to nonlinear MIPs, there has been much less emphasis on the nonlinear case among the research community. Some work in this direction is described in Borchers and Mitchell, 1994, Borchers and Mitchell, 1997, and Sahinidis, 1996, and elsewhere. (See the survey paper by Hansen et al., 1993.) Borchers and Mitchell, 1991, describe computational results with the use of an interior point branch-and-bound method for linear mixed integer programming problems, and Mitchell, 1996a, Mitchell, 1996b, and Mitchell et al., 1997, survey the use of interior point methods to solve integer programming and combinatorial optimization problems. Other methods for mixed integer nonlinear programming include methods based on Bender's decomposition (e.g., see Floudas, 1995), the outer approximation method of Duran and Grossman, 1986, Lagrangean decomposition and trust-region based approaches Michelon and Maculan, 1991, and Mauricio and Maculan, 1997. Nevertheless, the coupling of nonlinear solvers within the branch-and-bound framework is relatively unexplored.

The goal of this work is to incorporate the computational advances in nonlinear programming within the basic branch-and-bound framework to assist in tackling 0/1 mixed integer programs with nonlinearities appearing in either the objective or constraints. Thus, the general problem considered is of the form

$$\begin{array}{lrll} \text{minimize} & f(x) & & \\ \text{subject to:} & g(x) & \leq 0 & \\ & x_i & \in \{0,1\} & \forall\ i = 1, \ldots, p, \end{array} \tag{NMIP}$$

where $x \in \Re^n$, $g : \Re^n \rightarrow \Re^m$ and $p \leq n$. Without loss of generality, bounds on the variables are integrated into the constraints $g(x) \leq 0$.

In Section 2 we describe the nonlinear interior-point based solver we developed to solve the branch-and-bound subproblems. The integration of the nonlinear solver within the branch-and-bound tree is described in Section 3. In Section 4 we briefly describe the parallel implementation, and preliminary numerical results are reported in Section 5.

1.2 INTERIOR-POINT BASED NONLINEAR SOLVER

At each node of the branch-and-bound tree a subproblem — obtained by fixing certain 0/1 variables to zero or one, and relaxing the integral restrictions on the remaining 0/1 variables — must be solved. Thus, each subproblem is a nonlinear programming problem with only continuous variables. As such, it can be written as:

$$\begin{array}{lrl} \text{minimize} & f(x) & \\ \text{subject to:} & g(x) & \leq 0 \end{array} \tag{NLP}$$

There has been a tremendous amount of research among the nonlinear programming community on finding efficient algorithms for (NLP) (see, for example, Boggs et al., 1996, Boggs et al., 1994, Domich et al., 1991, El-Bakry et al., 1996, Murtagh and Saunders, 1978, McCormick, 1991, and Monteiro and Wright, 1992). In our current implementation, we employ the method of sequential quadratic programming (SQP) — which has been shown to be very effective in solving practical nonlinear programming problems — for solving (NLP). We note that when f and g are convex and differentiable, the SQP approaches return an optimal solution for (NLP) and thus a true lower bound for (NMIP). Suppose x^k is the current iterate. The direction d for computing the next iterate is obtained by first solving the quadratic problem:

$$\begin{array}{lrl} \text{minimize} & \nabla f(x^k)^T d + \frac{1}{2} d^T B(x^k) d & \\ \text{subject to:} & g(x^k) + \nabla g(x^k) d & \leq 0, \end{array} \tag{QP}$$

where $B(x^k)$ is an approximation of the hessian of the Lagrangian for (NLP), evaluated at x^k.

This quadratic problem is solved via an interior-point algorithm, an approach which has been shown to be computationally very effective by Lustig et al., 1992, and Vanderbei, 1994. The progress of the algorithm and the choice of a steplength is guided by a merit function. In our current implementation, we apply a simple L_1 merit function of the form:

$$f(x) + \rho||g^+(x)||_1 \tag{MF}$$

where ρ is a constant chosen to be greater than $||y||_\infty$ with y the dual multiplier of (QP); and $g_i^+(x)$ denotes the violation of constraint g_i when evaluated at x. Below we describe the interior-point solver implemented to solve problem (QP) and some special features of our code.

1.2.1 Overview of the Interior-point Solver.

Our interior point solver, based on the one developed by Boggs et al., 1996, and Domich et al., 1991, is developed to solve the quadratic problem (QP). For convenience of notation, we rewrite (QP) as

$$\begin{array}{ll} \text{minimize} & c^T d + \frac{1}{2} d^T Q d \\ \text{subject to:} & Ad \quad \leq b. \end{array} \tag{QP}$$

The first step of the solver is to preprocess the augmented matrix $[A, b]$ in order to remove redundant rows and columns, dominated rows and columns, as well as columns associated with fixed variables. If it is necessary to perform Phase I on (QP) to obtain an initial feasible solution, we modify (QP) to include an artificial variable with a dynamically altered cost. The iterates d^i are always strictly feasible in this modified problem, and the artificial variable is driven to zero as optimality is approached, provided the original (QP) is feasible. At the ith iterate, d^i, the next iterate d^{i+1} is determined by an (O3D) step as well as a Newton centering step. The (O3D) directions, v^1, v^2, v^3, chosen are the dual affine direction, the centering direction, and the order three correction direction. As optimality is approached, the centering direction is replaced by a direction that moves the iterate away from the constraint which is closest in the dual affine direction. Let $s > 0$ be the corresponding vector of slacks for (QP), and let D be the diagonal matrix with $D_{ii} = 1/s_i^2$. Then the three directions mentioned above are obtained by solving systems of the form

$$(A^T D A + \frac{1}{r_0} Q)\, v = \bar{v}$$

to find v. Here, $\bar{v}$ is an appropriately chosen vector and r_0 is an estimate of the objective function residual; that is, the gap between the objective value of (QP) evaluated at d^i and the optimal objective value. The steplengths taken in each of these directions are obtained by solving a three-dimensional subproblem:

$$\begin{array}{ll} \text{minimize} & \tilde{c}^T \tilde{x} + \frac{1}{2} \tilde{x}^T \tilde{Q} \tilde{x} \\ \text{subject to:} & \tilde{A}\tilde{x} \quad \leq \tilde{b} \end{array} \tag{O3D}$$

where $R = [v^1, v^2, v^3]$, $\tilde{c} = R^T(c + Qd^i)$, $\tilde{Q} = R^T Q R$, $\tilde{A} = AR$, $\tilde{b} = b - Ad^i$, and $\tilde{x} \in \Re^3$.

Clearly, the origin is feasible for (O3D). However, since there is no guarantee that an optimal solution exists, the O3D solver returns a vector $\tilde{x}^*$ that is either declared to be optimal, or tends to be very large in magnitude due to unboundedness of (O3D). In the case of optimality, we set $d^{i+1} = d^i + \alpha\gamma R\tilde{x}^*$, where $\alpha \in [0, 1]$, and γ is the safe steplength (i.e., the largest step possible without violating the constraints in (QP)). Otherwise, $\tilde{x}^*$ is scaled (to avoid taking too large of a step) before d^{i+1} is selected as above. We use a Newton centering step after each (O3D) step to center the iterate and to improve the rate of convergence. This centering routine also involves solving a three dimensional subproblem, and it is based on the one described in Boggs et al., 1996, and Domich et al., 1991. The stopping criteria for the interior-point solver are when the relative duality gap is less than or equal to the *optimality tolerance*; or when there is no relative improvement in the primal objective; or when there are no good steps obtained from the (O3D) routine and the Newton recentering step. In our implementation, initially the quadratic programming subproblems (QP) are solved using a relatively big optimality tolerance; then, as the solution to the underlying nonlinear programming problem is approached, the tolerance is decreased in order to achieve greater accuracy.

1.2.2 Special Features within the SQP Solver

We remark that the principal advantage of the implemented interior point algorithm over other interior point approaches to solving quadratic programming problems is that only one matrix, $A^T A + Q$, has to be factorized. This can be performed quite efficiently when the nonzero structures of $A^T A$ and Q share many entries. Most other methods (see, for example, Vanderbei, 1994) require the factorization of a matrix of the form

$$\begin{bmatrix} Q & A^T \\ A & G \end{bmatrix}$$

for an appropriately chosen diagonal matrix G. This latter factorization in general amounts to considerably more work than the one we use. However, our approach in general does not provide a dual feasible solution until termination, and it has not been proven to enjoy quadratic convergence.

Below, we highlight the iterative dual refinement procedure implemented to overcome the shortcomings of our approach, as well as some features which are implemented within our current SQP solver.

Scaling of input matrix. Before any numerical calculation, the interior-point solver first performs iterative scaling of the rows and columns of the constraint matrix of (QP). In particular, each row (column) is scaled by the geometric mean of the maximum and minimum of the absolute values of the nonzero entries of the row (column). We perform this iterative procedure until the ratio of the described maximum and minimum in each row (column) is within a certain pre-specified range. We also apply scaling within each O3D step.

Heuristic jumpstart of (QP). The interior-point solver includes a Phase I procedure to obtain a feasible interior starting point. Since Phase I involves calculations with large values, we also include an inexpensive heuristic for finding an initial interior starting point. The heuristic first sets the primal solution to zero. Then it determines the smallest right-hand-side coefficient. If this coefficient is positive, we scale it by a factor between 0 and 1. Otherwise, we set the value to -1.0. After that, each non-slack variable is assigned this value, scaled by the number of columns and the norm of the corresponding column in the constraint matrix. The slacks are then assigned values according to the current slack values. If the resulting vector is feasible and is in the interior of the constraint matrix, this point will be used as the warmstart for the interior-point solver. Otherwise, Phase I will be called.

Heuristic minimum ordering and diagonal augmentation. At the beginning of the (QP) solve, one minimum ordering is performed on the matrix $A^T A+Q$. We include a minimum degree ordering implemented as described in George and Liu, 1981, as well as a fast heuristic which orders the rows in nondecreasing order of the number of nonzero entries in each row. During numerical factorization, diagonal elements will be augmented whenever their values fall below a threshold zero tolerance. Within the O3D routines, the step $\tilde{x}$ is solved using a dense-Cholesky factorization with pivoting. We again augment diagonal elements when needed to ensure positive definiteness in the 3×3 matrix.

Iterative dual refinement. Since (O3D) is a primal method, it often does not provide a very accurate dual solution for (QP), especially if the algorithm is terminated prematurely. To improve the dual values of (QP), three dual refinement steps are performed:

> The first refinement, designed to reduce dual infeasibility, works within each primal iterate in the interior-point solver. If the estimated dual variable y for a primal iterate is infeasible, its value will be iteratively refined so as to reduce dual infeasibility. The second refinement occurs before the interior-point solver exits (QP). If the dual solution remains infeasible, it will be recalculated based on the corresponding theoretical values.
>
> We exploit the presence of simple bounds on the primal variables to increase the flexibility of selection of the values of the dual variables. For example, if one of the primal constraints is a simple bound on

a variable, then there is a slack variable in the corresponding dual constraint, and this variable can be adjusted to move towards feasibility, if necessary, or to improve the dual value.

We employ basis identification, discussed in El–Bakry et al., 1994, and Mitchell et al., 1997, which requires exploiting complementary slackness relationships to fix some dual variables at zero. In the quadratic case, this involves determining which constraints hold at equality, fixing some dual values to zero, and then searching for an appropriate dual solution.

Update dual variables of NLP. The dual solutions to (QP) can be used as dual solutions to (NLP). We use a slightly more conservative approach, and take the updated dual solution to (NLP) to be a weighted average of the former dual solution to (NLP) and the dual solution to (QP).

1.3 THE NONLINEAR BRANCH-AND-BOUND SOLVER

The SQP solver described above is embedded within a branch-and-bound framework. At the root node of the branch-and-bound tree, the solver begins by solving the nonlinear programming relaxation of (NMIP) to obtain a lower bound on the objective value. After the initial solve, the heuristic is called. If a feasible integer solution is found, the associated objective value serves as the current upper bound. (By default in a branch-and-bound algorithm, the initial upper bound is set to ∞, see Nemhauser and Wolsey, 1988, for details of the algorithm.) If the lower bound is less than the upper bound, we proceed with the branch-and-bound tree search. Throughout the tree search, whenever a node cannot be fathomed, a branching variable, based on either the smallest index among the fractional binary variables, or the most infeasible value (if there is a tie, the one with the smallest index is chosen), will be selected. Two new nodes, one with the selected variable set to 0, and the other with the selected variable set to 1, will be created and added to the set of active nodes. The order of processing nodes is based on the "best estimate" criterion. (Note that the bound on each node is only an estimate of its lower bound, due to our premature termination of the SQP solve as described below.) We now summarize some special features in our branch-and-bound solver.

Premature termination of SQP and QP solvers. At each node of the branch-and-bound tree, some binary variables are fixed to 0 or 1. We solve the corresponding nonlinear relaxation using the SQP approach, which uses the interior point algorithm (described in Section 2.1) to approximately solve the quadratic programming subproblems generated from the SQP method. The degree of premature termination of the SQP solver is controlled by selecting the optimality tolerance appropriately. Note that there are four possible outcomes at each node. In three of them, it suffices to solve the relaxation approximately.

If the optimal solution to the relaxation is fractional, it suffices to only solve the relaxation approximately, as basis identification techniques such as those used in El–Bakry et al., 1994, can be used to determine which variables are tending to fractional values.

If the optimal value to the relaxation is greater than the current best upper bound on the optimal value of the (NMIP), the relaxation only has to be solved accurately enough to determine this, and the node can then be fathomed.

If the relaxation is infeasible, it only has to be solved accurately enough to obtain a lower bound on the relaxation greater than the current best upper bound on the optimal value of the NMIP. This node can then be fathomed.

When the optimal solution is integral feasible and yields an objective which may result in updating of the best current upper bound, we solve the relaxation using a tightened optimality tolerance and cross-check to ensure integrality conditions are indeed satisfied by the solution.

The premature termination of the SQP and QP solvers help to reduce the iteration counts and avoid unnecessary computation, this is accomplished by allowing the optimality tolerance to range from 10^{-4} (for a loose optimality condition) to 10^{-13} (when accurate solution is important for correct conclusion).

Advanced warmstarts. In a branch-and-bound framework, it is desirable to exploit information about the parent node when solving the child node. This information should provide a good initial solution (or *warmstart*) for the child.

When the simplex algorithm is used to solve integer linear programming problems by branch-and-bound, it is advantageous to start the solution process at the child node from the optimal basis to the parent, reoptimizing using dual simplex. This is the fundamental idea of "advanced basis" in a simplex-based branch-and-bound solver. Typically, the set of indices of the basic variables of the optimal solution of each parent node is stored. Thus, for each active branch-and-bound node, a basis (corresponding to the optimal basis of its parent node) is stored for restart purpose.

The situation is much different with an interior point method. In this case, the optimal solution to the parent is not a good starting point for the child, since it is too close to the boundary. Typically, an interior point method will first move into the interior of the feasible region and then move towards the optimal solution to the child. It is better to start the interior point method at the child with a point that is more centered. Therefore, rather than storing a set of indices, it is desirable to store an interior point for the parent problem, and use this as a restart point for the child. Unfortunately, it is prohibitively expensive to store a double-precision vector of an initial interior point for each tree node. As the branch-and-bound tree grows, this will lead to memory explosion and prohibit further continuation of the solution process. However without a warmstart, the interior point algorithm may take an unacceptably large number of iterations before convergence is achieved.

Analogous to the "advanced basis," we introduce the idea of "advanced warmstart" in an interior-point based branch-and-bound solver. At the root node of the branch-and-bound tree, a collection of interior feasible points are reserved. As we select a node, we also select the most appropriate vector from the collection for warmstart. Typically we select one which is further away from the boundary of the feasible region. If it is not feasible to the subproblem, we perform an ad-hoc heuristic to try modifying this warmstart vector such that it becomes primal feasible. This collection of warmstarts is updated dynamically as the tree search proceeds.

Adaptive heuristic procedure and local warmstart updates. For each node at the depth of a multiple of 8, we perform a heuristic before branching. For difficult integer programming instances, the existence of good integer feasible solutions help to prune active nodes, and reduce the size of the branch-and-bound tree. This has a direct impact on the speedup of the solution process. We experimented with some primal heuristics. The current implementation involves an extension of the primal heuristics described in Bixby et al., 1996, and Bixby and Lee, 1998, which have been shown to be very effective for mixed 0/1 linear integer programming instances. The heuristic starts by using the warmstart associated with the current node. During successive solves within the heuristic, this vector is updated appropriately and, as such, serves as a local warmstart for the next solve. As is done when solving (SQP) within each branch-and-bound node, initially the heuristic is solved to a looser optimality tolerance within each consecutive solve. However, a tighter tolerance is employed when more variables are fixed and a higher degree of accuracy is desired.

Preprocessor. Preprocessing is performed on each branch-and-bound node. This procedure includes the commonly used techniques of removal of redundant rows, columns, dominated rows and columns as well as checks for infeasibility. After that, fixed columns are removed and the right-hand-side is updated appropriately. Next we scan for rows with exactly one nonzero entry. If the value for such a nonzero variable satisfies its bounds, the row is eliminated and the bounds for the variable are updated. Otherwise, the subproblem is declared infeasible.

A more involved type of preprocessing would be to use specialized techniques for finding upper and lower bounds, which may make it possible to prune a node more quickly. Techniques described in the literature include the use of Lagrangean decomposition; see, for example, Borchers and Mitchell, 1994, and Michelon and Maculan, 1991. We do not currently employ such techniques.

1.4 PARALLEL IMPLEMENTATION

Branch-and-bound is an inherently parallel algorithm. Indeed, there has been consistent interest among both the computer science and the integer programming research communities in designing efficient parallel implementations, see for example Applegate et al., 1994, Bixby et al., 1996, Cannon and Hoffman, 1990, Eckstein, 1994, and Gendron and Crainic, 1994. Due to the "NP-hard" nature of integer programs, it is to be expected that an efficient parallel branch-and-bound algorithm will be able to solve some difficult integer programs which would otherwise be intractable running on sequential codes. Indeed, three of the most competitive commercial MIP solvers (CPLEX, OSL, XPRESS), include a parallel branch-and-bound implementation on certain computer platforms. One distinct difference of our parallel branch-and-bound is its capability of solving *nonlinear* mixed 0/1 integer programs.

Our parallel implementation uses TreadMarks[1] as the parallel platform. TreadMarks is a *distributed shared memory* (DSM) system for networks of Unix workstations and distributed-memory multiprocessors, such as the IBM SP2. DSM enables processes running on different workstations to share data through a network-wide virtual memory, even though the hardware provided by the network lacks the capability for one workstation to access another workstation's physical memory as discussed in Bixby et al., 1996, Keleher et al., 1994, and Li and Hudak, 1989. Our current implementation is done on a loosely coupled network of SUN SPARC20/M61 workstations, connected via 10BaseT ethernet cable.

At the initial phase of the parallel code, one processor is responsible for reading in the problem. That processor also solves the initial nonlinear programming relaxation. If the optimal solution is integral feasible, the algorithm is done; otherwise, the heuristic is called. After that, sequential branch-and-bound is performed to spawn enough active nodes until the accumulated number of nodes exceeds a predetermined threshold. At this point, parallel execution is activated. The global data in our implementation consists of the best upper bound (for minimization problems), its corresponding solution, the global list of active nodes, and a collection of warmstart vectors (generated at the root node). For an individual processor, the initial setup amounts to reading in a copy of the nonlinear programming relaxation, as well as all the modifications to it after performing preprocessing and scaling at the root. The tree search is handled in a "pseudo" centralized manner. Based on best-estimate selection, an idle processor fetches a short list of active nodes from the global list, reads the current best upper bound and the collection of warmstart vectors. One node from the short list is selected, and the associated nonlinear program is solved. If an integral solution is obtained, this node is fathomed without further branching; otherwise, a local heuristic is called according to the heuristic interval setting. If the heuristic is performed and a better upper bound is obtained, the best upper bound and solution are "updated" locally within this processor. If there is no gap between the nonlinear programming objective value and this upper bound, this node is fathomed; otherwise, a branching variable is selected and two new nodes corresponding to the selected variable are created. The processor keeps all this new global data locally, and continues the solution process on another node from its short collection until all of the nodes are exhausted. After that, if an updated best upper bound is available, or if there are new active nodes created, or if warmstart vectors have been modified, all this information will lead to global update via the lock and release mechanisms on the global data. The above process is repeated until the entire list of active nodes is exhausted and every processor is idle, signaling the completion of the parallel processing.

1.5 NUMERICAL RESULTS

Gathering our experience in the implementation of interior point solvers and branch-and-bound algorithms (including Bixby et al., 1996, Bixby et al., 1995, Bixby and Lee, 1998, Borchers and Mitchell, 1991, Borchers and Mitchell, 1994, Borchers and Mitchell, 1997, Lee, 1997b, Lee, 1997a, Mitchell, 1996a, Mitchell, 1996b, and Mitchell, 1997), the entire optimization code is built "in-house" in C to allow greater flexibility and efficiency in adapting features from the SQP solver within the tree search environment. In this section, we report our preliminary tests performed on five portfolio problems and 16 mixed 0/1 instances from MIPLIB. These instances arise from real applications, and the experiments will serve as our initial stage of reporting empirical performance of the solver as a quadratic MIP solver. The tests were performed on a network of SUN SPARC/M61(50MHZ) workstations.

Table 1.1 shows the problem statistics and the performance of the branch-and-bound solver on 16 MIPLIB instances. *Name*, *Rows*, *Cols*, and *0/1 var.* correspond to the name of the problem instance, and the number of rows, columns and 0/1 variables in the constraint matrix. *LP Obj.*, *MIP Obj.*, *BB nodes*, and *CPU Time (secs)* indicate the objective for the initial LP relaxation, the optimal objective for the integer program, the number of branch-and-bound nodes solved, and the CPU time elapsed.

Table 1.1 Problem Descriptions and Branch-and-Bound Statistics of MIPLIB Instances

Name	Rows	Cols	0/1 var.	Initial LP Obj.	Optimal MIP Obj.	BB nodes	CPU Time (secs)
air01	23	771	771	6743.0	6796	4	62.3
bm23	20	27	27	20.57	34	101	328.2
egout	98	141	55	149.589	568.1007	634	1290.2
fixnet3	478	878	378	40717.018	51845	131	972.6
fixnet4	479	878	378	4257.966	8922	376	722.8
fixnet6	479	878	378	1200.884	3981	60	120.5
lp4l	85	1086	1086	2942.5	2967	41	395.6
misc01	54	83	82	57.0	563.5	49	235.5
misc02	39	59	58	1010.0	1690	14	13.0
mod008	6	319	319	290.931	307	194	576.2
mod013	62	96	48	256.016	280.95	120	305.7
p0033	16	33	33	2520.57	3089	62	85.2
p0040	23	40	40	61796.55	62027	12	20.3
rgn	24	180	100	48.799	82.199	15	139.7
stein15	13	9	15	7.0	9	23	34.2
stein27	118	27	27	13.0	18	620	543.2

The portfolio problem deals with the decision of selecting a portfolio consisting of at most K out of n potential investments. Here, the investor has a fixed amount of capital to invest, and must decide how much of this capital should go to each investment. For each dollar invested in investment i, the return at time T is a_i. The investor desires to select a portfolio that will provide a return of at least M at time T. The optimization concerns minimizing the total variance in the investment so that the investor is guaranteed a net return of at least M. The portfolio problem can be modeled as a quadratic 0/1 integer program of the following form:

$$
\begin{array}{lrrll}
\text{minimize} & x^T Q x & & & \\
\text{subject to:} & x_i & -y_i & \leq 0 & \forall\ i = 1, \ldots, n \\
 & & \sum_{i=1}^{n} y_i & \leq K & \\
 & \sum_{i=1}^{n} a_i x_i & & \geq M & \\
 & \sum_{i=1}^{n} x_i & & = 1 & \\
 & & y_i & \in \{0, 1\} & \forall\ i = 1, \ldots, n \\
 & & x_i & \geq 0 & \forall\ i = 1, \ldots, n
\end{array}
$$

where Q is a dense positive definite matrix representing the covariance between different investments.

In Table 1.2, the column labels are analogous to those in Table 1.1. Here, *NLP Obj.* corresponds to the objective value of the initial nonlinear programming relaxation.

To investigate the importance of the idea of "advanced warmstart" within the tree search environment, we performed branch-and-bound on the problem instances with only one parameter change: advanced warmstart flag. Table 1.3 compares the average interior-point iterations per branch-and-bound node solved, with and without advanced warmstart vectors. The first two columns correspond to the total CPU time, and the average number of interior point iterations per branch-and-bound nodes when advanced warmstarts are employed within the tree search environment. The last two columns indicate the corresponding values when advanced warmstart is turned off. With the advanced warmstarts, the reduction in solution times in all

Table 1.2 Problem Descriptions and Branch-and-Bound Statistics for 5 Portfolio Problems

Name	Rows	Cols	0/1 var.	Initial NLP Obj.	Optimal MIP Obj.	BB nodes	CPU Time (secs)
port150	755	300	150	1.49695	1.496950	49	902.6
port100	505	200	100	1.3735	1.37357	29	495.3
port50	255	100	50	1.77815	1.83227	30	90.5
port50b	255	100	50	1.54860	1.548633	78	180.7
port10	55	20	10	2.89533	2.89533	12	19.2

reported instances range from 1.17 to as much as 7.8 times, with an average 2.5 times decrease in the number of interior point iterations solved per branch-and-bound node. These empirical results provide solid evidence that using advanced warmstarts within the tree search environment can dramatically reduce both the number of interior point iterations performed, and the overall solution time.

Table 1.3 Effect of "advanced warmstart"

	Warmstart		No Warmstart	
Name	CPU Time (secs)	Aver. ip iters	CPU Time (secs)	Aver. ip iters
port150	902.6	7	2459.4	21
port100	495.3	6	1100.3	20
port50	90.5	6	350.7	18
port50b	180.7	5	310.2	12
port10	19.2	6	29.4	8
air01	62.3	17	189.2	42
bm23	328.2	7	510.5	15
egout	1290.2	7	3012.3	18
fixnet3	972.6	8	3742.5	30
fixnet4	722.8	12	3015.2	37
fixnet6	120.5	8	936.2	39
lp4l	395.6	7	1073.1	19
misc01	235.5	5	412.2	11
misc02	13.0	6	40.2	14
mod008	576.2	7	1320.1	16
mod013	305.7	5	810.2	11
p0033	85.2	7	211.2	15
p0040	20.3	6	31.2	8
rgn	139.7	7	402.1	21
stein15	34.2	4	40.3	5
stein27	543.2	5	792.2	8

We next report the speedup of our solver over four distributed machines in Table 1.4. Let T_n denote the time elapsed when n processors are used. To capture the idle time spent on the machines due to communication overhead during the parallel process, in our tests, T_n was always measured using "wall-clock" time, and was recorded starting from reading in the problem instance to the final shutdown of all processors after printing the solution. We define the *speedup* for n processors to be the ratio $\frac{T_1}{T_n}$.

We observe that problems with sequential running time greater than 100 seconds enjoyed fairly decent speedup. However, few of the problems with sequential solution times under 100 seconds achieved significant

Table 1.4 Speedup on n SPARC20/M61

Name	$\frac{T_1}{T_2}$	$\frac{T_1}{T_3}$	$\frac{T_1}{T_4}$
port150	1.94	2.76	3.60
port100	1.90	2.80	3.41
port50	1.67	2.41	2.23
port50b	1.78	2.45	3.22
port10	1.34	1.21	1.90
air01	1.12	1.23	1.02
bm23	1.87	2.87	3.56
egout	2.02	2.97	3.87
fixnet3	2.12	3.09	3.99
fixnet4	1.96	2.78	3.50
fixnet6	1.75	2.50	3.0
lp4l	1.70	2.61	3.2
misc01	1.56	1.98	2.33
misc02	1.23	1.20	1.01
mod008	1.98	2.67	3.76
mod013	1.78	2.77	3.65
p0033	1.77	2.56	3.41
p0040	1.43	1.21	1.12
rgn	1.57	1.54	1.24
stein15	1.34	1.17	1.23
stein27	1.90	2.78	3.56

speedup. Indeed, in some instances, the parallel runnings times are almost the same as the sequential time. Such behavior is not unexpected, and can be largely attributed to communication overhead. In addition, several of these models simply do not generate enough nodes to justify (or necessitate) the use of parallelism. We will continue to explore the capability of our solver on different classes of nonlinear 0/1 mixed integer programs. We expect the parallel code will provide good speedup on some difficult instances.

1.6 CONCLUSIONS AND CURRENT RESEARCH

The preliminary results on the nonlinear branch-and-bound solver described herein are quite encouraging. Empirical results justify our idea of using advanced warmstarts within each branch-and-bound node. In addition, we remark that from the results of our numerical tests, we observed that the dual refinement aids in reducing iteration counts within the interior-point solver. An interior point method is good at getting close to optimality fast, thus it is a perfect candidate to solve the relaxations in a branch-and-bound framework for nonlinear mixed integer programming problems. The challenge lies in effective exploitation of the nonlinear solver, and the branch-and-bound algorithm so as to adapt them efficiently into a single unified optimization tool. Further numerical tests will be conducted on different classes of nonlinear mixed integer programs arising in real applications to further evaluate the effectiveness of the SQP solver within a branch-and-bound framework. As pointed out in section 2, when f and g are convex and differentiable, our current algorithm solves the NMIP instances to proved optimality. Present algorithmic work includes improving the robustness and capability of the solver (e.g., to handle nonconvex objective and constraint functions). The solution of nonconvex problems will obviously require the use of a method for generating reliable lower bounds on the global optimum of the nonlinear programming relaxation of the problem. In addition, the near-optimality solution obtained in each subproblem can serve as the guide for generating cuts.

Notes

1. TreadMarks is a trademark of Parallel Tools, L.L.C.

References

[Applegate et al., 1994] Applegate, D., Bixby, R., Chvátal, V., and Cook, W. (1994). The traveling salesman problem. Technical report, DIMACS, Rutgers University, New Brunswick, NJ.

[Bixby et al., 1995] Bixby, R. E., Cook, W., Cox, A., and Lee, E. K. (1995). Parallel mixed integer programming. Technical Report CRPC–TR95554, Department of Computational and Applied Mathematics, Rice University, Houston, Texas.

[Bixby et al., 1996] Bixby, R. E., Cook, W., Cox, A., and Lee, E. K. (1996). Computational experience with parallel mixed integer programming in a distributed environment. Technical report, Department of Computational and Applied Mathematics, Rice University, Houston, Texas. To appear in *Annals of Operations Research, Special Issue on Parallel Optimization*.

[Bixby and Lee, 1998] Bixby, R. E. and Lee, E. K. (1998). Solving a truck dispatching scheduling problem using branch-and-cut. *Operations Research*, 46:355–367.

[Boggs et al., 1996] Boggs, P. T., Domich, P. D., and Rogers, J. E. (1996). An interior-point method for general large scale quadratic programming problems. *Annals of Operations Research*, 62:419–438.

[Boggs et al., 1994] Boggs, P. T., Tolle, J. W., and Kearsley, A. J. (1994). A practical algorithm for general large scale nonlinear optimization problems. Technical report, National Institute of Standards and Technology, Gaithersburg, MD 20899.

[Borchers and Mitchell, 1991] Borchers, B. and Mitchell, J. E. (1991). Using an interior point method in a branch and bound algorithm for integer programming. Technical Report 195, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180. Revised July 7, 1992.

[Borchers and Mitchell, 1994] Borchers, B. and Mitchell, J. E. (1994). An improved branch and bound algorithm for mixed integer nonlinear programming. *Computers and Operations Research*, 21(4):359–367.

[Borchers and Mitchell, 1997] Borchers, B. and Mitchell, J. E. (1997). A computational comparison of branch and bound and outer approximation methods for 0-1 mixed integer nonlinear programs. *Computers and Operations Research*, 24:699–701.

[Cannon and Hoffman, 1990] Cannon, T. L. and Hoffman, K. L. (1990). Large-scaled 0/1 linear programming on distributed workstations. *Annals of Operations Research*, 22:181–217.

[Domich et al., 1991] Domich, P. D., Boggs, P. T., Rogers, J. E., and Witzgall, C. (1991). Optimizing over three-dimensional subspaces in an interior-point method for linear programming. *Linear Algebra and its Applications*, 152:315–342.

[Duran and Grossman, 1986] Duran, M. A. and Grossman, I. E. (1986). An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339.

[Eckstein, 1994] Eckstein, J. (1994). Parallel branch-and-bound algorithms for general mixed integer programming on the CM–5. *SIAM Journal on Optimization*, 4:794–814.

[El–Bakry et al., 1994] El–Bakry, A. S., Tapia, R. A., and Zhang, Y. (1994). A study of indicators for identifying zero variables in interior–point methods. *SIAM Review*, 36:45–72.

[El-Bakry et al., 1996] El-Bakry, A. S., Tapia, R. A., Tsuchiya, T., and Zhang, Y. (1996). On the formulation and theory of the primal-dual newton interior-point method for nonlinear programming. *Journal of Optimization Theory and Applications*, 89:507–541.

[Floudas, 1995] Floudas, C. A. (1995). *Nonlinear and Mixed Integer Optimization*. Oxford University Press.

[Gendron and Crainic, 1994] Gendron, B. and Crainic, T. G. (1994). Parallel branch-and-bound algorithms: survey and synthesis. *Operations Research*, 42:1042–1066.

[George and Liu, 1981] George, J. A. and Liu, J. W. H. (1981). *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ.

[Hansen et al., 1993] Hansen, P., Jaumard, B., and Mathon, V. (1993). Constrained nonlinear 0–1 programming. *ORSA Journal on Computing*, 5:97–119.

[Keleher et al., 1994] Keleher, P., Cox, A., Dwarkadas, S., and Zwaenepoel, W. (1994). TreadMarks: Distributed memory on standard workstations and operating systems. In *Proceedings of the 1994 Winter Usenix Conference*, pages 115–131.

[Lee, 1997a] Lee, E. K. (1997a). A branch-and-cut approach to treatment plan optimization for permanent prostate implants. Technical report, Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

[Lee, 1997b] Lee, E. K. (1997b). Computational experience of a general purpose mixed 0/1 integer programming solver (mipsol). Technical report, Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

[Li and Hudak, 1989] Li, K. and Hudak, P. (1989). Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 4:229–239.

[Lustig et al., 1992] Lustig, I. J., Marsten, R. E., and Shanno, D. F. (1992). On implementing Mehrotra's predictor-corrector interior point method for linear programming. *SIAM Journal on Optimization*, 2:435–449.

[Mauricio and Maculan, 1997] Mauricio, D. and Maculan, N. (1997). A trust region method for zero-one nonlinear programming. *RAIRO — Operations Research*, 31:331–341.

[McCormick, 1991] McCormick, G. P. (1991). The superlinear convergence of a nonlinear primal-dual algorithm. Technical Report T–550/91, School of Engineering and Applied Science, George Washington University, Washington, D.C.

[Michelon and Maculan, 1991] Michelon, P. and Maculan, N. (1991). Lagrangean decomposition for integer nonlinear programming with linear constraints. *Mathematical Programming*, 52:303–313.

[Mitchell, 1996a] Mitchell, J. E. (1996a). Interior point algorithms for integer programming. In Beasley, J. E., editor, *Advances in Linear and Integer Programming*, chapter 6, pages 223–248. Oxford University Press.

[Mitchell, 1996b] Mitchell, J. E. (1996b). Interior point methods for combinatorial optimization. In Terlaky, T., editor, *Interior Point Methods in Mathematical Programming*, chapter 11, pages 417–466. Kluwer Academic Publishers.

[Mitchell, 1997] Mitchell, J. E. (1997). Computational experience with an interior point cutting plane algorithm. Technical report, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180–3590. Revised: April 1997.

[Mitchell et al., 1997] Mitchell, J. E., Pardalos, P. P., and Resende, M. G. C. (1997). Interior point methods for combinatorial optimization. Technical report, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180. Accepted for publication in *Handbook of Combinatorial Optimization*, 1998.

[Monteiro and Wright, 1992] Monteiro, R. D. C. and Wright, S. J. (1992). A globally and superlinearly convergent potential reduction interior point method for convex programming. Technical Report SIE 92–13, SIE Department, University of Arizona, Tucson, AZ.

[Murtagh and Saunders, 1978] Murtagh, B. A. and Saunders, M. A. (1978). MINOS 5.5 user's guide. Technical Report SOL 83–20, SOL, Stanford University, Palo Alto, CA. Revised: July 1998.

[Nemhauser and Wolsey, 1988] Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. John Wiley, New York.

[Sahinidis, 1996] Sahinidis, N. (1996). BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205.

[Vanderbei, 1994] Vanderbei, R. J. (1994). LOQO: An interior point code for quadratic programming. Technical report, Statistics and Operations Research, Princeton University, Princeton, NJ 08544. To appear in *Optimization Methods and Software*.