Numerical experience with lower bounds for MIQP branch–and–bound *

Roger Fletcher and Sven Leyffer†
University of Dundee

3^{rd} October 1995

Abstract

The solution of convex Mixed Integer Quadratic Programming (MIQP) problems with a general branch–and–bound framework is considered. It is shown how lower bounds can be computed efficiently during the branch–and–bound process. Improved lower bounds such as the ones derived in this paper can reduce the number of QP problems that have to be solved. The branch–and–bound approach is also shown to be superior to other approaches to solving MIQP problems. Numerical experience is presented which supports these conclusions.

Key words: Integer Programming, Mixed Integer Quadratic Programming, Branch–and–Bound

AMS subject classification: 90C10, 90C11, 90C20

1 Introduction

One of the most successful methods for solving mixed–integer nonlinear problems is branch–and–bound. Land and Doig [16] first introduced a branch–and–bound algorithm for the travelling salesman problem. Dakin [3] introduced the now common branching dichotomy and was the first to realize that it is possible to solve Mixed Integer Nonlinear Programming problems by branch–and–bound. This paper presents a general framework algorithm for convex Mixed–Integer Quadratic Programming (MIQP) branch–and–bound and shows, in particular, how lower bounds can be computed efficiently.

The introduction of MIQP master programs into Outer Approximation [7] has motivated us to study the solution of MIQP problems in detail. However, MIQP problems are not only interesting as master problems in the aforementioned algorithm but also in their own right, having various applications such as index tracking for passive portfolio management [15], the optimal sizing and siting of substations in a network routing problem and so on.

Several authors suggest branching rules in the context of mixed–integer linear programming problems (e.g. [1], [22], [2], [20]) and some of these rules have been adopted to the mixed–integer nonlinear case (c.f. [12]). However, despite their potential importance no attempt has been made to derive lower bounds for the MIQP branch–and–bound algorithm. This paper suggests a procedure which provides lower bounds that can be used

*This work was supported by SERC grant number SERC GR/F 07972

†University of Dundee, Department of Mathematics, Dundee, DD1 4HN, Scotland, U.K., sleyffer@mcs.dundee.ac.uk

in branch–and–bound. Our numerical experience suggests that improved lower bounds can reduce the number of QP problems solved considerably.

Branch–and–bound is not the only approach to solving MIQP problems. In a recent paper, Lazimy [17] shows how Generalized Benders Decomposition can be employed to solve MIQP problems via a finite sequence of QP–subproblems and MILP master programs. Flippo and Rinnoy Kan [8] improve Lazimy's approach so that it becomes applicable to a wider class of MIQP problems. Alternatively, Outer Approximation (Duran and Grossmann [4]) or LP/QP based branch–and–bound (Quesada and Grossmann [21]) can readily be adapted ([18]) to solve MIQP problems. However, all these approaches are essentially based upon linearizations and consequently ignore higher order terms. Our own experience, presented in Section 5 suggests that they are inferior to branch–and–bound. A review of different approaches to MIQP problems is also given by Volkovich et. al. [19].

The present paper is organized in six sections. In Section 2 the notation and terminology required to describe our MIQP branch-and–bound algorithm is introduced. The general method to derive improved lower bounds is presented in Section 3, and it is shown how these lower bounds can be obtained efficiently. A small example is presented in Section 4 together with some numerical results for the branch–and–bound solver. In Section 5 results for the other MIQP solvers are presented and compared to the branch–and–bound solver. The conclusions are summarized in Section 6.

2 Notation and terminology

Branch–and–bound is a general framework for solving integer and combinatorial problems. The combinatorial part of the problem (determining the optimal integer assignment) is solved by a tree search in which QP relaxations of the MIQP problem are solved and non–integer QP–solutions are eliminated by adding simple bounds (branching). By using lower and upper bounds on the optimal objective value it is possible to limit the tree search, thus avoiding complete enumeration.

The branch–and–bound methodology for solving MIQP problems is very similar to Beale's [1] paper on MILP branch–and–bound. The particular problem which is considered here is

$$\mathcal{P} \left\{ \begin{array}{ll} \min\limits_{x} & f(x) = \frac{1}{2} x^T G x + g^T x \\ \text{subject to} & A^T x \geq b \\ & x \in X \ , \ x_i \text{ integer } \forall \ i \in I \end{array} \right.$$

where f is a convex function and the set X represents simple bounds on the variables. It is assumed that the feasible region is bounded, which can be achieved for instance by ensuring that X is bounded. There is no difficulty in extending the methods of this paper to solve problems where some of the "integer" variables are restricted to take a set of discrete values other than just the integers.

In order to describe a branch–and–bound algorithm it is necessary to introduce some notation and terminology. Let $\mathcal{P}'$ denote the problem obtained from $\mathcal{P}$ by relaxing all integer restrictions. Problem $\mathcal{P}'$ is then an ordinary QP problem. The branch–and–bound algorithm starts by solving $\mathcal{P}'$, giving x' as its solution. If x' satisfies all integer restrictions it is said to be an *integer feasible solution* and in this case also solves $\mathcal{P}$ and the algorithm stops. Otherwise there exists a variable x'_j, $j \in I$ which is not integer (and is said to be *fractional*). Branch–and–bound then proceeds by *branching* on a fractional variable, x'_j say. This is done by introducing two new subproblems obtained from $\mathcal{P}'$ by adding the simple bound $x_j \geq [x'_j + 1]$ to one and $x_j \leq [x'_j]$ to the other, where $[a]$ denotes the largest integer not greater than a. The solution of $\mathcal{P}$ is contained in the feasible region of one of the two new subproblems and the process can be repeated.

The Branch–and–bound algorithm continues to solve and generate new subproblems performing a tree search where the nodes of the tree correspond to QP subproblems. The nodes obtained by branching are called *child–nodes* and the node from which they are generated is called the *parent–node*. A node which has been fully explored is referred to as being *fathomed*. A *pending node* is a node which has been generated by branching but has not yet been solved. The branch–and–bound algorithm searches the tree until no pending nodes remain.

It is not always necessary to search the complete tree and the success of branch–and–bound is partly due to the fact that whole subtrees can be excluded from further consideration whilst fathoming their respective root node. A node has been fathomed if it is recognized as infeasible or produces an integer feasible solution. In the latter case the optimal value of the node also provides an *upper bound* on the solution of $\mathcal{P}$. This upper bound is then used to fathom nodes whose optimal value or lower bound is greater or equal than the current upper bound. The last point indicates that the existence of easy–to–compute lower bounds on the value of a node can reduce the number of problems that need to be solved. For more details we refer the reader to [9, Chapter 10], [23, page 171] and [12].

In the remainder of this section we briefly comment on a number of implementation details that are left open by a general description of the branch–and–bound algorithm. The first question concerns the choice of QP solver. Using a dual method to solve the subproblems offers the most straightforward way to exploit the structure which the branching introduces into the problem. Branching makes the solution of the parent problem infeasible, but a dual feasible point is readily available from the dual solution of the parent problem. A dual active set method could therefore take immediate advantage of a feasible starting point. Moreover, since a dual Active Set Method is an ascend method it can make use of an upper bound as a cut–off value terminating a QP solve prematurely. The present implementation of an MIQP branch–and–bound algorithm uses a primal active set method with six different degrees of warm starts to solve the QP subproblems. The warm starts enable the QP solver to recycle information from the solve of the parent problem. This solver has the additional advantage that it implements a technique to resolve degeneracy and thus gives guaranteed termination even in the presence of round–off errors [6]. This guarantee contributes to the robustness of the MIQP code.

Another question is that concerning the choice of the pending node to be solved next. This is resolved in favour of a depth–first–search of the tree with the additional feature that backtracking is done to the most promising node. The aim is to quickly find an integer feasible solution so that the resulting upper bound can be used to fathom nodes whose lower bounds exceed the new upper bound.

Finally, the important question regarding the choice of the branching variable is discussed. Many authors suggest a branching rule based on the importance of the integer variables (see [1] for MILP and [12] for MINLP). This rule exploits user–defined priorities which are assigned to the integer variables, and branches on the most important variable first.

For variables of equal priority tie-breaking rules are required and these are discussed next. A very simple branching rule which has proved successful both for MINLP branch–and–bound [12] and the Travelling Salesman Problem [20] is based on the amount by which the variables violate the integer restrictions. This strategy selects the integer variable which is furthest from its nearest integer value and is referred to as *maximal fractional part branching* by Breu and Burdet [2]. The aim is to obtain the largest increase in the objective function so that more nodes can be pruned later on.

Another branching rule which does not require any problem specific knowledge but is based on pseudo–costs has been introduced by Benichou et.al. [22]. They compute

so–called pseudo–costs which estimate the change in the objective induced by the branching. Gupta and Ravindran report numerical experience with pseudo–costs for a range of MINLP test problems and conclude that it is inferior to the branching rule which is described in the previous paragraph.

Körner [14] proposes a branching rule which aims to minimize the size of the tree so that fewer QP problems need be solved. He shows how the tree size resulting from a particular branching order can be estimated efficiently. The drawback of these estimates is that they are based on a special tree–structure. Instead of branching (on x_i) by creating two problems, a range of problems is created with fixed values $x_i = i \; \forall \; i = l_i, l_i+1, \ldots, u_i$. This branching structure can lead to wide trees especially if the simple bounds are not very strong.

3 Improved lower bounds for MIQP branch–and–bound

A lower bound for both child nodes is readily available as the optimal value of the parent problem, since $\mathcal{P}$ is convex. However, these bounds are usually very weak and rarely give rise to any fathoming at a later stage once an integer feasible solution has been obtained. Improved lower bounds have therefore been considered in [1] for MILP and are readily derived in the context of an Active Set Method (ASM) (e.g. [5]) for MILP problems.

This section considers MIQP problems and shows how improved lower bounds can be derived for the child nodes of $\mathcal{P}'$, where $\mathcal{P}'$ is the QP relaxation of $\mathcal{P}$ as defined in Section 2. The procedure can be readily applied to any other node of the MIQP branch–and–bound tree. Let the solution to $\mathcal{P}'$ be x' and assume that $x'_j \; , \; j \in I$ is fractional. Define $\mathcal{P}^-$ as the QP obtained from $\mathcal{P}'$ by branching down (i.e. by adding the simple bound $x_j \leq [x'_j]$) and $\mathcal{P}^+$ as the QP obtained from $\mathcal{P}'$ by branching up (i.e. by adding $x_j \geq [x'_j + 1]$).

The derivation of lower bounds is best explained using a tableau representation of the dual ASM (e.g. [6]). This tableau is *implicitly* available after the parent QP has been solved with any ASM–like QP solver. In an active set method the constraints are divided into two disjoint sets, the *active set* $\mathcal{A}$ and the set $\mathcal{I}$ of inactive constraints. If no degeneracy occurs, then the active set $\mathcal{A}$ can be identified with the set of active constraints. Partitioning the multipliers λ and the residual r of the constraints (including the simple bounds) accordingly, the tableau then defines basic or dependent variables $\lambda_{\mathcal{A}}$ and $r_{\mathcal{I}}$ in terms of independent or nonbasic variables $r_{\mathcal{A}}$ and $\lambda_{\mathcal{I}}$.

In general then, the corresponding LCP tableau has the form

$$\begin{array}{c} \\ \lambda_{\mathcal{A}} \\ r_{\mathcal{I}} \end{array} \begin{array}{c} \begin{array}{cc} r_{\mathcal{A}} & \lambda_{\mathcal{I}} \end{array} \\ \left[\begin{array}{cc} G' & A' \\ -A'^T & H' \end{array} \right] \end{array} \left[\begin{array}{c} \lambda'_{\mathcal{A}} \\ r'_{\mathcal{I}} \end{array} \right] \tag{1}$$

where the current value of the basic variables is given by the right hand side $(\lambda'_{\mathcal{A}}, r'_{\mathcal{I}})$, and the current value of the nonbasic variables is zero. Columns in the tableau indicate how the basic variables are affected by changes in the nonbasic variables. It is possible to derive expressions for A', G' and H' and these are given towards the end of this section in the case that the QP solver employs a null–space method.

First it is described how the dual ASM can be used to derive improved lower bounds for the case that less than n constraints are active at the solution of the parent problem $\mathcal{P}'$. Any dual feasible point for $\mathcal{P}^-$ or $\mathcal{P}^+$ would provide a lower bound on the optimum of $\mathcal{P}^-$ or $\mathcal{P}^+$. The solution of the parent problem $\mathcal{P}'$ is dual feasible in both $\mathcal{P}^-$ and $\mathcal{P}^+$. However, the lower bound derived from this point rarely gives rise to much fathoming and we therefore wish to improve this lower bound.

In order to improve the lower bound on $\mathcal{P}^-$ and $\mathcal{P}^+$ derived from the optimal solution of the parent problem $\mathcal{P}'$, we consider one step of the dual ASM. Problems $\mathcal{P}^-$ and $\mathcal{P}^+$ differ from $\mathcal{P}'$ by the addition of a single simple bound which makes the primal solution of $\mathcal{P}'$ infeasible in $\mathcal{P}^-$ and $\mathcal{P}^+$. The method that is described here can be seen as moving parametrically from x' towards $[x'_j]$ (or $[x'_j + 1]$) whilst maintaining dual feasibility. The move is completed when a pivot would be required in the dual ASM, as we wish to avoid the expense of this computation.

The dual ASM of Goldfarb and Idnani [11] starts with a dual feasible point. This makes the method very amenable to MIQP branch–and–bound since upon solving the parent problem its solution (x', λ') is dual feasible in both child problems. The dual ASM takes one constraint that is violated and tries to increase its multiplier. As a consequence of the tableau structure the increase of the multiplier reduces the primal infeasibility whilst maintaining all other feasibilities.

All necessary quantities for the step can be computed from the final LCP tableau of the parent problem. This tableau is either available directly or indirectly (as in our case) if the parent problem is solved with an ASM. Assume that the final LCP tableau of the parent problem is given by (1). Since x'_j is fractional, it follows that $j \in \mathcal{I}$, or in other words x'_j is not at a simple bound. Let λ_j be the complementary variable of the simple bound constraint on x_j. The dual ASM aims to increase λ_j away from zero until one of the other multipliers becomes zero. This corresponds to moving the bound of x_j parametrically towards its new value, $[x'_j]$ say. The aim is to determine how far the bound can be moved in the null–space of the active constraints without destroying dual feasibility.

The effect of increasing λ_j on the multiplier of the active constraints is determined by the column of the LCP tableau (1) corresponding to λ_j. In particular the effect on the multipliers of the active constraints is

$$\lambda_{\mathcal{A}} = \lambda'_{\mathcal{A}} - \alpha a'_j$$

for moving x_j towards $[x'_j + 1]$ and

$$\lambda_{\mathcal{A}} = \lambda'_{\mathcal{A}} + \alpha a'_j$$

for moving x_j towards $[x'_j]$, where a'_j is the column of A' corresponding to λ_j. Also λ_j itself is increased from 0 to α. The maximum step α that can be taken without losing dual feasibility is determined by the dual ratio test. Thus

$$\alpha^+ = \min_{i\,:\,a'_{ij}>0} \left(\frac{\lambda'_i}{a'_{ij}} \right)$$

gives the dual feasible steplength for moving towards $[x'_j + 1]$ and

$$\alpha^- = \min_{i\,:\,a'_{ij}<0} \left(-\frac{\lambda'_i}{a'_{ij}} \right)$$

gives the dual feasible steplength in the second case. If there exists no $a'_{ij} > 0$ or no $a'_{ij} < 0$ respectively then α is set to ∞, since the dual is unbounded. In this case it follows that the primal is infeasible and the respective dual node is therefore fathomed. This particular child problem is therefore not added to the list of pending problems. Similarly, if the improved lower bound is greater than or equal to the current upper bound, the child problem is not added to the list of pending problems.

Finally the lower bounds on $\mathcal{P}^+$ and $\mathcal{P}^-$ can be computed. Let s be the search direction of the dual ASM, i.e. $s_i = [h'_j]_i$ if $i \in \mathcal{I}$, $i \leq n$ and $s_i = 0$ otherwise, where h'_j is column

j of H'. If $\phi_j = x_j - [x_j]$ is the fractional part of x_j and if f' is the optimal value of the parent then the lower bounds L^+ and L^- respectively are given by the values of the Lagrangian at the new points,

$$L^+ = f' + \tfrac{1}{2}(\alpha^+)^2 s^T G s + \alpha^+(1 - \phi_j - \alpha^+ h'_{jj})$$

$$L^- = f' + \tfrac{1}{2}(\alpha^-)^2 s^T G s + \alpha^-(\phi_j - \alpha^- h'_{jj}).$$

There are two terms contributing to the improved bounds. First there is the quadratic term corresponding to the second order change in the objective. The lower bound is further improved by the last term which corresponds to a Lagrangian type penalty for the violation of the new simple bound, namely the product of the multiplier α of the new simple bound with the amount by which the simple bound is not yet satisfied.

In the special case that n constraints are active at the solution of $\mathcal{P}'$ then the final LCP tableau can be simplified to

$$\begin{array}{c} \\ \lambda_{\mathcal{A}} \\ r_{\mathcal{I}} \end{array} \begin{array}{c} \begin{array}{cc} r_{\mathcal{A}} & \lambda_{\mathcal{I}} \end{array} \\ \begin{bmatrix} G' & A' \\ -A'^T & 0 \end{bmatrix} \end{array} \begin{bmatrix} \lambda'_{\mathcal{A}} \\ r'_{\mathcal{I}} \end{bmatrix}$$

where A' is a (nonsingular) $n \times n$ matrix. In this case $h'_j = 0$ and the lower bounds are $L^+ = f' + \alpha^+(1 - \phi_j)$ for $\mathcal{P}^+$ and $L^- = f' + \alpha^- \phi_j$ for $\mathcal{P}^-$. In the case where n constraints are active at the solution of the QP it is possible to derive the same lower bounds using marginal costs p_j^+ and p_j^- and the convexity of f similar to [5, p. 337].

In the remainder of this section we indicate how the lower bound procedure can be implemented and give an operation count. The QP solver employed in the present implementation uses a primal active set method, where equality QP problems are solved by a null space method [6].

At the end of each solve a partition of the constraint matrix A (which now includes also the simple bounds) into active and inactive constraints is available. However, similar information is likely to be available with other solvers, like for instance with solvers based on Linear Complementarity Problems. It is then possible to derive expressions for G', A' and H' ([6] Theorem 2.1).

- $A' = T^T A_{\mathcal{I}}$ and the off–diagonal blocks in (1) are skew–symmetric,
- $G' = U$ is symmetric and negative definite if G is positive definite,
- $H' = -A_{\mathcal{I}}^T H A_{\mathcal{I}}$ is symmetric and negative semi definite.

The matrices H, T and U are defined through the inverse

$$\begin{bmatrix} G & -A_{\mathcal{A}} \\ -A_{\mathcal{A}}^T & 0 \end{bmatrix}^{-1} = \begin{bmatrix} H & -T \\ -T^T & U \end{bmatrix}. \tag{2}$$

At the end of the QP solve of the parent problem we assume that LU factors of the basis matrix $[A_{\mathcal{A}} : V]$ are available, where $A_{\mathcal{A}}$ is the matrix whose columns are the active constraint normals and V is any matrix such that $[A_{\mathcal{A}} : V]$ is nonsingular. This includes QR factors of $A_{\mathcal{A}}$ but also the important practical case where V is made up of columns of the unit matrix. If

$$[A_{\mathcal{A}} : V]^{-1} = \begin{bmatrix} Y^T \\ Z^T \end{bmatrix}$$

then it is possible to obtain expressions for H, T and U (e.g. [5]). These lead to explicit expressions for A' and H', given by

$$H' = -A_{\mathcal{I}}^T Z (Z^T G Z)^{-1} Z^T A_{\mathcal{I}}$$
$$A' = (Y - Z(Z^T G Z)^{-1} Z^T G Y)^T A_{\mathcal{I}} \ ,$$

where the "reduced Hessian" $(Z^T G Z)$ is available *implicitly* through LDL^T factors at the end of the solve of the parent QP. Note that it is not necessary to form $(Z^T G Z)$ explicitly. Instead we update factors of the reduced Hessian matrix as the basis changes.

Thus it is possible to derive an expression for a'_j

$$a'_j = A' e_j = Y^T \underbrace{A_{\mathcal{I}} e_j}_{e_j} - Y^T G \, \underbrace{Z \, \underbrace{(Z^T G Z)^{-1} \, \underbrace{Z^T \underbrace{A_{\mathcal{I}} e_j}_{e_j}}_{v}}_{w}}_{u} .$$

A similar expression can be derived for $h'_j = H' e_j$. Note that the intermediate vector u is used both for h'_j and a'_j.

In the case where fewer than n constraints are active at the solution of $\mathcal{P}'$ the following computations are carried out to derive the lower bound. We also give the flop count in the case that dense matrix factorizations are employed (here $k = \dim(\mathrm{null}(A_{\mathcal{A}}))$) .

1. Form $v = Z^T e_j$ by performing 1 solve with the LU factors of $[A_{\mathcal{A}} : V]^{-1}$. ($n^2$ flops)
2. Use LDL^T factors of the reduced Hessian to form $w = (Z^T G Z)^{-1} v$. (k^2 flops)
3. Form $u = Zw$ using the LU factors of $[A_{\mathcal{A}} : V]$. (n^2 flops)
4. Compute m scalar products to form $h'_j = -A_{\mathcal{I}}^T u$. ($mn$ flops)
5. Form the matrix vector product $u = e_j - Gu$. (n^2 flops)
6. Form $a'_j = Y^T u$, using the LU factors of $[A_{\mathcal{A}} : V]$. (n^2 flops)
7. Carry out the ratio test to find α^+ and α^-. ($n + 2$ flops)
8. Compute the lower bounds L^+ and L^-. ($n^2 + n + 4$ flops)

Summing the individual operation counts shows that the lower bounds can be computed in $5n^2 + mn + k^2 + \mathcal{O}(n)$ flops. Note that the operation count for step 4 is overly pessimistic. Many inactive constraints will usually be simple bounds for which the scalar product in 4 costs only 1 and not n flops. It is worth mentioning that sparse matrix techniques for factoring and for representing $[A_{\mathcal{A}} : V]$ can significantly reduce the n^2 component in the operation count. In particular, if V is made up of unit columns this operation count can be reduced to $\mathcal{O}((n-k)^2)$ flops. We do not include the cost for forming LU factors in the fop count since these factors are available if the parent QP has been solved by an ASM.

We are grateful to an anonymous referee for pointing out that if range space factors were used to solve the parent problem, then the computation of a'_j in Step 6 would be cheaper. However, range space factors are less convenient for handling upper and lower bounds on the variables in a QP solver and the factors of the Hessian matrix G may be expensive to compute or not even exist.

The procedure to compute a lower bound is considerably cheaper than a QP solve, but an order of magnitude more expensive than the simpler branching rules based upon the most fractional variable. Only extensive numerical testing can reveal whether or not this additional effort results in an overall reduction in CPU time for MIQP problems and this together with a small illustrative example is considered in the next section.

4 A small example and numerical experience

It is instructive to consider applying the above procedure to a simple example:

$$\mathcal{P}\left\{\begin{array}{ll}\min\limits_{x} & x_1^2 - x_1x_2 + x_2^2 - 3x_1 \\ \text{subject to} & -x_1 - x_2 \geq -2 \\ & x \geq 0\ ,\ x_1 \text{ integer}\end{array}\right.$$

The solution of $\mathcal{P}$ with its integer restrictions relaxed is $x' = (\frac{3}{2}, \frac{1}{2})$, $f' = -\frac{99}{36}$ and the final LCP tableau is given by

$$\begin{array}{c} \\ \lambda_1 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} r_1 & \pi_1 & \pi_2 \end{array} \\ \left[\begin{array}{c|cc} -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \hline \frac{1}{2} & -\frac{1}{6} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} & -\frac{1}{6} \end{array}\right] \end{array} \left[\begin{array}{c} \frac{1}{2} \\ \frac{3}{2} \\ \frac{1}{2} \end{array}\right]$$

where π_i denotes the complementary variable to x_i. Let $\mathcal{P}^-$ denote the problem obtained from $\mathcal{P}$ by adding the new upper bound $x_1 \leq 1$. To compute a lower bound on the solution of $\mathcal{P}^-$ the dual ratio test is carried out, giving $\alpha^- = \min\left(-\frac{1}{2}/-\frac{1}{2}\right) = 1$. The search direction in terms of the x variables is $s = (-\frac{1}{6}, \frac{1}{6})^T$ and the step of the dual active set method moves to the point $x^- = x' + \alpha^- h' = \left(\frac{4}{3}\ ,\ \frac{2}{3}\right)^T$ which results in the improved lower bound of $L^- = -\frac{84}{36}$. This improvement corresponds to closing 80% of the gap between the solution values of $\mathcal{P}$ and $\mathcal{P}^-$ (which is $-\frac{81}{36}$). Similarly the lower bound for $\mathcal{P}^+$ can be computed as $L^+ = -2$. The step of the dual ASM is illustrated in Figure 1.

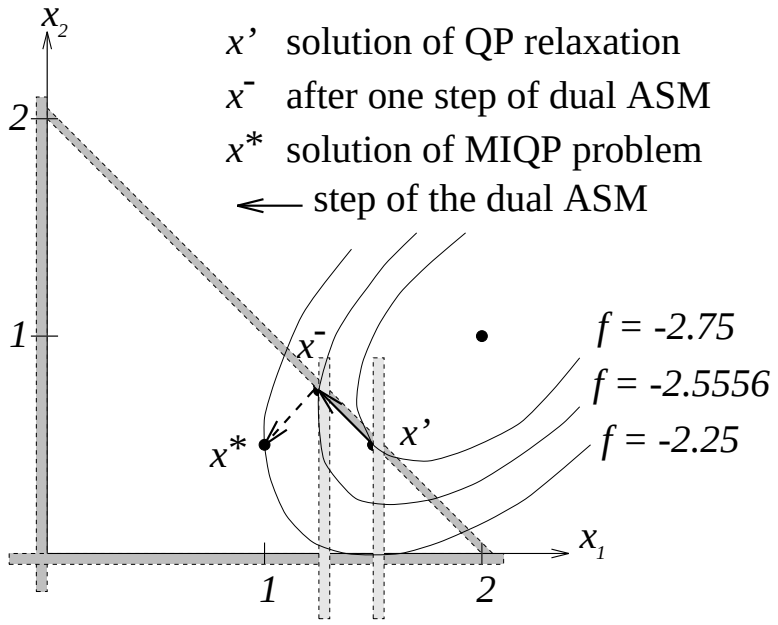

Figure 1: Illustration of the dual active set method.

It is indicated in Section 3 that improved lower bounds are not cheaply computed and it is therefore important to carry out numerical testing to assess to what extent the savings in the number of problems solved are mirrored by savings in the CPU time. To

this end three different branching rules have been implemented which make use of the improved lower bounds in different ways. BR1 branches on the most fractional integer variable first and uses no improved lower bounds at all. BR2 computes lower bounds L_j^+ and L_j^- for all fractional integer variables and branches on the integer variable that satisfies

$$\max_j \min(L_j^+, L_j^-).$$

Finally, BR3 is a hybrid of BR1 and BR2, designed to expose the effect of the new bounds. It uses branching rule BR1 to determine the branching variable, but in addition computes lower bounds for the two child problems. Each branching rule requires a different number of floating point operations every time it is applied. BR1 requires $\mathcal{O}(n)$, BR3 requires $\mathcal{O}(n^2)$ and BR2 requires $\mathcal{O}(n^3)$ flops per application. An alternative to BR3 is to solve the child problems by the dual ASM and use a cut–off value to terminate the solve (BR3 is in fact equivalent to one dual ASM iteration).

The test problems come from a variety of backgrounds. "Beale" and "HS76" are problems found in the literature (e.g. [13]). The problems "AVGAS", "AFIRO" and "SHARE2B" are originally LP problems which can be found in e.g. [24] and the SOL test set respectively. These LP problems are modified by adding integer restrictions and by replacing the linear objective function by a quadratic objective with positive definite Hessian matrix. QAP(6,4) is a quadratic assignment problem for allocating 4 plants to 6 locations. Finally, "portfolio 1" and "portfolio 2" are problems that arise in the index tracking for passive portfolio management. In both cases a portfolio of 1 and 2 investments out of 17 possible investments is selected.

Table 1 gives a brief description of the test problems. The first column gives the name of the problem followed by four columns that detail the dimensions of the problems; n_i is the number of integer variables, n_c is the number of continuous variables, m_e gives the number of equality constraints and finally m_i details the number of inequality constraints. f' is the optimal value of the QP relaxation and f^* is the optimal value of $\mathcal{P}$. The last column gives the range of the integer variables x_I.

Problem	n_i	n_c	m_e	m_i	f'	f^*	Integer Range
Beale	3	0	0	1	-8.8889	-8.0	$0 \leq x_I \leq 5$
HS76	4	0	0	3	-4.6818	-4.5	$0 \leq x_I \leq 5$
AVGAS (1a)	8	0	0	10	-4.6319	-1.5	$0 \leq x_I \leq 1$
AVGAS (1b)	8	0	0	10	-4.4832	-1.0	$0 \leq x_I \leq 1$
AFIRO (1a)	26	6	8	19	1048.01	1205.5	$0 \leq x_I \leq 5$
AFIRO (1b)	26	6	8	19	1081.14	1205.5	$0 \leq x_I \leq 5$
AFIRO (2a)	19	13	8	19	1048.01	1068.2	$0 \leq x_I \leq 5$
AFIRO (2b)	19	13	8	19	1074.52	1099.5	$0 \leq x_I \leq 5$
AFIRO (2c)	19	13	8	19	64.21	66.96	$0 \leq x_I \leq 5$
SHARE2B (8)	8	71	12	84	2977.67	3132.6	$0 \leq x_I \leq 5$
SHARE2B (12)	12	71	12	84	2977.67	3159.7	$0 \leq x_I \leq 5$
SHARE2B (24)	24	59	12	84	2977.67	3332.8	$0 \leq x_I \leq 5$
QAP(6,4)	24	55	4	6	$6.21 \cdot 10^{-3}$	3.0	$0 \leq x_I \leq 1$
portfolio 1	17	17	2	17	$-8.59 \cdot 10^{-4}$	$-7.11 \cdot 10^{-4}$	$0 \leq x_I \leq 1$
portfolio 2	17	17	2	17	$-8.59 \cdot 10^{-4}$	$-7.92 \cdot 10^{-4}$	$0 \leq x_I \leq 1$

Table 1: Description of MIQP Test Problems

The following tables give the results of the test run on a SUN-SPARC SLC using the FORTRAN 77 compiler with the –fast option. All problems are solved in single precision FORTRAN to a "mixed accuracy" of $\epsilon = 10^{-6}$; that is an approximate solution $\hat{x}$ is found

for which $|f(\hat{x}) - f^*| \leq (1 + |f^*|) \cdot \epsilon$. The QP problem at each node is solved using `bqpd`, which implements a primal ASM and resolves degeneracy giving a guaranteed termination even in the presence of round–off errors [6]. Table 2 gives the number of QP problems generated by each of the branching rules, the number of QP problems solved and the percentage of problems solved. The CPU time needed by each branching rule and for the three other solvers is presented in Table 4 in the next section. Runs which took more than 2,000 seconds CPU time were abandoned.

Problem	BR1			BR2			BR3		
	generated	solved	% solved	generated	solved	% solved	generated	solved	% solved
Beale	7	7	100	7	4	57	7	4	57
HS76	5	5	100	5	3	60	5	3	60
AVGAS (1a)	21	20	95	19	14	74	20	13	65
AVGAS (1b)	19	19	100	29	17	59	19	13	68
AFIRO (2a)	9	9	100	11	7	64	8	5	63
AFIRO (2b)	17	17	100	17	13	76	14	11	79
AFIRO (2c)	39	35	90	31	19	61	45	30	67
AFIRO (1a)	3,439	2,948	86	17	15	88	3,180	2,225	70
AFIRO (1b)	1,125	900	80	29	27	93	985	684	69
SHARE2B (8)	65	57	88	15	9	60	65	45	69
SHARE2B (12)	729	708	97	49	29	59	645	396	61
SHARE2B (24)	$>$ 2000 s CPU			304	185	61	$>$ 2000 s CPU		
QAP(6,4)	319	319	100	397	353	89	319	289	91
portfolio 1	33	26	79	31	26	84	33	25	76
portfolio 2	171	168	98	209	180	86	173	136	79

Table 2: Number of continuous problems generated and solved by branch–and–bound

BR3 and BR1 use the same branching rule. Therefore, comparing their respective results in Table 2 shows for BR1 that very little pruning takes place if the lower bounds are taken as the optimal value of the parent problem. BR3 in comparison solves a lower percentage of the problems that are generated. The difference in the number of problems generated for the larger examples can easily be explained. The effect of the improved pruning for BR3 is that both branching rules search the tree in a slightly different order. As one would expect, the lower bounds result only in a modest improvement in terms of the number of QP problems solved (this would be similar if the QP problems were solved with a dual ASM and a cut–off value). Note that one cannot expect to prune more than 50% of the nodes through lower bounds, which limits the scope for improvement through pruning for the hybrid branching rule BR3.

An interesting feature is that there exist difficult MIQP problems (such as AFIRO and SHARE2B) where BR2 improves the performance of the branch–and–bound scheme dramatically. With BR2 only a fraction of the number of QPs that are solved using BR1 or BR3 are solved. This reduction in the number of QPs translates into a reduction in terms of CPU time resulting in an order of magnitude reduction in the CPU time of BR2 compared to the other two branching rules for these problems.

This improvement appears to be due to the ability of BR2 to find an integer feasible solution faster than the two other methods. For example for AFIRO (1a) BR2 finds the optimal solution after only 15 QP solves whereas BR1 and BR3 require 2,348 and 2,143 QP solves respectively before the first integer feasible solution is found. Effectively, BR2

explores the neighbourhood of the solution of the parent problem in the direction in which branching is possible and this seems to improve the decision process that selects the next branching variable in a dramatic manner.

However, this observation appears to be true only for MIQP problems for which the notion of "neighbourhood" has a sensible meaning such as general MIQP problems. In particular, this is not true for 0–1 QPs and this explains the rather disappointing performance of BR2 on "AVGAS" and especially "QAP(6,4)" and "portfolio". For the latter two problems BR2 solves about 10% more QP problems which results in a doubling of the CPU times compared to BR1 and BR3 (which are the cheaper branching rules).

5 Numerical Experience with other solvers

In this section we report on experience which we obtained with other MIQP solvers. The solvers we implemented are based on Generalized Benders' Decomposition (GBD) (e.g. [10], [17] and [8]), Outer Approximation (OA) (e.g. [4], [7]) and LP/QP based branch–and–bound ([21]). All these solvers have in common that they solve the MIQP problem via a finite sequence of MILP master program relaxations and QP subproblems obtained by fixing the integer variables. The master program is obtained by projection on to the integer variables, together with dualization (for GBD) and outer approximation for OA. The LP/QP based branch–and–bound algorithm differs in the way in which the related master programs are solved. Instead of solving a sequence of successive MILP master program relaxations as for OA, the branch–and–bound process for solving the MILP master program is modified. Whenever an integer assignment is generated during the tree–search the corresponding QP subproblem is solved and new outer approximations are added to all pending LP problems. Thus instead of re–solving the master program, it is effectively updated. All three solvers were again implemented in single precision FORTRAN 77 using `bqpd` to solve the QP subproblems and an MILP version of the MIQP branch–and–bound solver of the previous section to solve the master problems. In order to gain a fair comparison the best branching rule for each problem was used in the tests.

Problem	OA		GBD		LP/QP	
	iterations	LPs solved	iterations	LPs solved	iterations	LPs solved
Beale	7	47	7	41	6	20
HS76	2	14	6	38	2	12
AVGAS (a)	10	195	13	145	10	42
AVGAS (b)	11	190	15	151	11	47
AFIRO (2a)	12	846	22	975	23	236
AFIRO (2b)	14	858	24	748	18	145
AFIRO (2c)	69	9,826	64	10,054	31	203
AFIRO (1a)	4	2,353	16	1,566	4	429
AFIRO (1b)	5	548	24	10,865	4	26
SHARE2B (8)	10	463	34	1,202	13	255
SHARE2B (12)	> 2000 s CPU		97	21,217	> 2000 s CPU	
SHARE2B (24)	> 2000 s CPU		> 2000 s CPU		> 2000 s CPU	
QAP(6,4)	361	9,116	373	8,976	361	935
portfolio 1	18	299	20	227	18	50
portfolio 2	94	8,159	122	5,952	98	356

Table 3: Number of iterations and number of LPs solved by OA, GBD and LP/QP

Table 3 shows the performance of GBD, OA and LP/QP based branch–and–bound on the test problems. The table shows the number of iterations (or QP solves) and the number of LP solves for each problem.

A comparison of the three other solvers with branch–and–bound shows that branch–and–bound is the clear winner both in terms of the number of problems solved and in terms of CPU time. With few exceptions branch–and–bound is an order of magnitude faster than any of the other solvers (and never slower). Moreover, the three relaxation based solvers failed to solve a number of problems within the time limit of 2,000 seconds CPU time. The reason for the poor performance of GBD, OA and LP/QP based branch–and–bound is that while they solve fewer QP problems the number of LP problems grows dramatically.

Problem	BR1	BR2	BR3	OA	GBD	LP/QP
Beale	0.00	0.01	0.02	0.16	0.14	0.07
HS76	0.01	0.02	0.01	0.06	0.15	0.06
AVGAS (1a)	0.16	0.16	0.14	2.42	1.26	0.59
AVGAS (1b)	0.14	0.17	0.14	2.26	1.33	0.63
AFIRO (2a)	0.69	0.71	0.49	113.19	49.42	69.45
AFIRO (2b)	0.90	1.02	0.70	118.98	36.92	52.55
AFIRO (2c)	2.64	2.06	2.36	836.30	362.64	30.02
AFIRO (1a)	112.49	1.70	92.79	126.73	49.21	35.64
AFIRO (1b)	32.60	3.19	28.56	31.69	440.77	4.09
SHARE2B (8)	37.54	13.09	31.76	470.93	125.33	907.43
SHARE2B (12)	418.70	29.96	238.87	$>$ 2000.00	1420.74	$>$ 2000.00
SHARE2B (24)	$>$2000.00	207.29	$>$2000.00	$>$ 2000.00	$>$ 2000.00	$>$ 2000.00
QAP(6,4)	9.87	15.54	9.63	1159.01	1529.23	139.96
portfolio 1	1.26	2.76	1.29	24.77	7.21	8.32
portfolio 2	9.58	18.27	8.15	$>$ 2000.00	234.82	69.87

Table 4: CPU times [seconds] for branch–and–bound, OA, GBD and LP/QP

Finally, LP/QP based branch–and–bound is seen to be more efficient than OA or GBD since it avoids the re–solution of successive MILP master programs and solves only a few more QP problems. GBD performs best on problems which have relatively few integer variables since its master problem involves only the integer variables (plus one dummy continuous variable). However, none of the three alternatives to branch–and–bound seems to be able to compete with branch–and–bound which is often an order of magnitude faster than any of the alternatives.

It should not come as a surprise that none of the solvers presented in this section is comparable to branch–and–bound. The reason for this is that the solution of a QP child problem is very cheap (the average number of pivots required to solve a child problem is between 1 and 2 for "AFIRO" and between 4 and 10 for "SHARE2B") resulting in an inexpensive tree–search (comparable to the MILP situation). However, this situation does not hold for MINLP branch–and–bound where each node may require the solution of several QP problems. Our own experience [18] shows that there exist important classes of MINLP problems for which branch–and–bound is not the best method.

6 Conclusions

It has been shown how improved lower bounds for MIQP branch–and–bound can be computed by parametrically taking one step of the dual active set method. A modest

improvement through the use of improved lower bounds for a standard branching rule is observed. An interesting point is that on some hard problems a branching rule based entirely on the new bounds achieved an improvement of an order of magnitude compared to a standard branching rule. The branch–and–bound solver has also been compared to three alternative solvers based on relaxing an equivalent MILP master program. The numerical tests show that these solvers are an order of magnitude slower than branch–and–bound.

References

[1] E.M.L. Beale. Integer programming. In D.A.H. Jacobs, editor, *The State of the Art in Numerical Analysis*, London, 1978. Academic Press.

[2] R. Breu and C.-A. Burdet. Branch–and–bound experiments in 0–1 programming. *Mathematical Programming Study*, 2:1–50, 1974.

[3] R.J. Dakin. A tree search algorithm for mixed integer programming problems. *Computer Journal*, 8:250–255, 1965.

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

[5] R. Fletcher. *Practical Methods of Optimization, 2^{nd} edition.* John Wiley, Chichester, 1987.

[6] R. Fletcher. Resolving degeneracy in quadratic programming. *Annals of Operations Research*, 47:307–334, 1993.

[7] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

[8] O.E. Flippo and A.H.G. Rinnoy Kan. A note on benders decomposition in mixed–integer quadratic programming. *Operations Research Letters*, 9:81–83, 1990.

[9] R.S. Garfinkel and G.L. Nemhauser. *Integer Programming.* John Wiley, New York, 1972.

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

[11] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27:1–33, 1983.

[12] O.K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

[13] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes.* Springer–Verlag, Berlin, 1981.

[14] F. Körner. A new branching rule for the branch–and–bound algorithm for solving nonlinear integer programming problems. *BIT*, 28:701–708, 1988.

[15] J.W. Kwiatowski. Algorithms for index tracking. Working Paper 90.1, University of Edinburgh, Centre for Financial Markets Research, 1990.

[16] A.H. Land and A.G Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[17] R. Lazimy. Improved algorithm for mixed-integer quadratic programs and a computational study. *Mathematical Programming*, 32:100–113, 1985.

[18] S. Leyffer. *Deterministic Methods for Mixed Integer Nonlinear Programming.* PhD thesis, University of Dundee, Dundee, Scotland, UK, December 1993.

[19] V.A. Roshchin O.V. Volkovich and I.V. Sergienko. Models and methods of solution of quadratic integer programming problems. *Cybernetics*, 23:289–305, 1987.

[20] M. Padberg and G. Rinaldi. A branch–and–cut algorithm for the resolution of large-scale symmetric travelling salesman problems. *SIAM Review*, 33:40–100, 1991.

[21] I. Quesada and I.E. Grossmann. An LP/NLP based branch–and–bound algorithm for convex MINLP optimization problems. *Computers and Chemical Engineering*, 16:937–947, 1992.

[22] M. Benichou J.M. Gauthier P. Girodet G. Hentges G. Ribiere and O. Vincent. Experiments in mixed–integer linear programming. *Mathematical Programming*, 1:76–94, 1971.

[23] H.A. Taha. *Integer Programming – Theory, Applications and Computations.* Academic Press, London, 1975.

[24] A.C. Williams. Computational experience with ellipsoid algorithms for linear programming. *Mathematical Programming Society, COAL Newsletter*, 5:37–48, 1981.