SOLVING QUADRATIC ASSIGNMENT PROBLEMS USING CONVEX QUADRATIC PROGRAMMING RELAXATIONS

Nathan W. Brixius

Dept. of Computer Science, University of Iowa, Iowa City, IA 52242

Kurt M. Anstreicher

Dept. of Management Sciences, University of Iowa, Iowa City, IA 52242

We describe a branch-and-bound algorithm for the quadratic assignment problem (QAP) that uses a convex quadratic programming (QP) relaxation to obtain a bound at each node. The QP subproblems are approximately solved using the Frank-Wolfe algorithm, which in this case requires the solution of a linear assignment problem on each iteration. Our branching strategy makes extensive use of dual information associated with the QP subproblems. We obtain state-of-the-art computational results on large benchmark QAPs.

KEY WORDS: Quadratic assignment problem, branch-and-bound, quadratic programming, Frank-Wolfe algorithm

1 INTRODUCTION

The quadratic assignment problem (QAP) in "Koopmans-Beckmann" form is

$$\begin{aligned} \mathrm{QAP}(A,B,C): \quad & \min \quad \mathrm{tr}(AXB+C)X^T \\ & \text{s.t.} \quad X \in \Pi, \end{aligned}$$

where A, B and C are $n \times n$ matrices, tr denotes the trace of a matrix, and Π is the set of $n \times n$ permutation matrices. Throughout we assume that A and B are symmetric. In a typical location application a_{ij} is the flow between facilities i and j, b_{kl} is the distance between locations k and l, and c_{ik} is the fixed cost of assigning facility i to location k.

It is well known that $\mathrm{QAP}(A,B,C)$ is an NP-Hard problem; for example combinatorial optimization problems such as traveling salesman and graph partitioning can be formulated as QAPs. Moreover QAPs have proven to be extremely difficult to solve to optimality in practice. General problems with $n = 20$ are challenging, and several problems with $n = 30$ have been open for as long as 30 years. For recent

surveys which include the history and applications of the QAP and a discussion of solution methods see [7], [10], and [31].

The usual approach to optimally solving QAP(A, B, C) is to employ a branch-and-bound (B&B) algorithm. (See [24] and [30] for alternatives based on polyhedral theory.) Early papers reporting results of B&B algorithms for QAP include [5], [8], [13], and [29]. The most important element in the construction of a B&B algorithm for the QAP appears to be the method used to obtain a bound for the subproblem (itself a lower-dimensional QAP) at each node of the B&B tree. Most B&B algorithms for the QAP have utilized the well-known Gilmore-Lawler bound (GLB). In [32] the GLB and a related "variance reduction" bound are used in a B&B algorithm to solve to optimality a variety of test problems up to size $n = 20$. Using a B&B algorithm based on GLB, and high-performance computing hardware, Clausen and co-workers [6, 12] solved to optimality problems up to size $n = 32$, the most difficult being the nug22 problem. (All problem names, such as nug22, are taken from QAPLIB [9].) The larger nug25 problem was subsequently solved [28] using a bounding approach based on dynamic programming.

Other bounds for the QAP include bounds based on a linear programming (LP) relaxation of the problem, and bounds based on eigenvalues of A and B. As shown in [25] a number of known bounds for QAP, including the GLB, can be obtained from feasible solutions for the dual of an LP relaxation. The best published B&B results using such dual-LP bounds are due to Hahn et al. [19]. In recent work [20] the approach of [19] is refined and used to obtain the best results to date on the difficult nug24 and nug25 problems, as well as the first solution of the kra30a problem.

Eigenvalue-based bounds for the QAP are described in [14] and [21]. Results in [11] show that the "projected eigenvalue bound" PB(A, B, C) of [21] may be competitive with GLB in a B&B algorithm. Results in [11] on the applicability of the "triangle decomposition bound" of [26] are much less encouraging.

A new bound for QAP(A, B, C) based on convex quadratic programming (QP) is described in [2]. The new bound, QPB(A, B, C) is related to the projected eigenvalue bound PB(A, B, C). The construction of QPB(A, B, C) utilizes a semidefinite programming (SDP) representation of the basic eigenvalue bound of [14], from [4]. In this paper we consider a complete B&B algorithm for the QAP based on the use of the QP bound at each node of the B&B tree. The lower bound QPB(A, B, C) is reviewed in Section 2. In Section 2 we also decribe an approach based on the Frank-Wolfe (FW) algorithm that we use to approximately solve the convex quadratic program associated with QPB(A, B, C). The FW algorithm generates dual information that can be used to estimate the effect of fixing an assignment $x_{ij} = 1$ to create a "child" problem at a node in the B&B tree. Our branching rules, described in Section 3, make extensive use of this dual information. In Section 4 we give computational results on a variety of problems of size $n > 15$ from QAPLIB. We obtain state-of-the-art results on many problems, including instances of the famous nugxx problems up to size $n = 24$.

Notation. We use $\operatorname{tr} A$ to denote the trace of a square matrix A, and $A \bullet B = \operatorname{tr}(AB^T)$. For symmetric matrices A and B we use $B \succeq A$ to denote that $B - A$

is positive semidefinite, and $B \succ A$ to denote that $B - A$ is positive definite. We use e to denote a vector of arbitrary dimension with each component equal to one. The Kronecker product of matrices A and B is denoted $A \otimes B$. For an $n \times n$ matrix X, $\mathbf{vec}(X)$ denotes the vector in $\Re^{n^2}$ obtained by stacking the columns of X atop one another, in the natural order. See [17] or [22] for basic properties of Kronecker products and $\mathbf{vec}(\cdot)$. For an $n \times n$ symmetric matrix A, $\lambda(A) \in \Re^n$ denotes the vector of eigenvalues of A.

The "minimal product" of two vectors x and y in $\Re^n$ is denoted $\langle x, y \rangle_-$, and is defined by

$$\langle x, y \rangle_- = \min_{\pi} \sum_{i=1}^{n} x_i y_{\pi(i)},$$

where $\pi(\cdot)$ is a permutation of $1, 2, \ldots, n$. It is easy to show that if $x_1 \leq x_2 \leq \ldots \leq x_n$, and $y_1 \geq y_2 \geq \ldots \geq y_n$, then $\langle x, y \rangle_- = x^T y$.

Throughout the paper we use the convention of letting the name of an optimization problem, such as $\mathrm{QAP}(A, B, C)$, also refer to the solution value of the problem.

2 THE QUADRATIC PROGRAMMING BOUND

We use the convex QP bound for $\mathrm{QAP}(A, B, C)$ described in [2]. Let V be an $n \times (n-1)$ matrix with orthonormal columns such that $e^T V = 0$, and define $\hat{A} = V^T A V$, $\hat{B} = V^T B V$. The bound is defined as

$$\begin{array}{lll} \mathrm{QPB}(A, B, C): & \min & \mathbf{vec}(X)^T Q\, \mathbf{vec}(X) + C \bullet X + \langle \lambda(\hat{A}), \lambda(\hat{B}) \rangle_- \\ & \text{s.t.} & Xe = X^T e = e \\ & & X \geq 0, \end{array}$$

where Q is a matrix of the form $Q = (B \otimes A) - (I \otimes S) - (T \otimes I)$. The matrices S and T are of the form $S = V\hat{S}V^T$, $T = V\hat{T}V^T$, where $\hat{S}$ and $\hat{T}$ are optimal solutions of the semidefinite programming problem

$$\begin{array}{lll} \mathrm{SDD}(\hat{A}, \hat{B}): & \max & \mathrm{tr}\, \hat{S} + \mathrm{tr}\, \hat{T} \\ & \text{s.t.} & (I \otimes \hat{S}) + (\hat{T} \otimes I) \preceq (\hat{B} \otimes \hat{A}). \end{array}$$

It is known [4] that $\mathrm{SDD}(\hat{A}, \hat{B}) = \langle \lambda(\hat{A}), \lambda(\hat{B}) \rangle_-$, and as shown in [2, Section 4] optimal $\hat{S}$, $\hat{T}$ can easily be obtained from the spectral decompositions of $\hat{A}$ and $\hat{B}$.

The quadratic programming bound $\mathrm{QPB}(A, B, C)$ is related to the projected eigenvalue bound [21]

$$\mathrm{PB}(A, B, C) = \langle \lambda(\hat{A}), \lambda(\hat{B}) \rangle_- + \mathrm{LAP}(D) - \frac{1}{n^2}(e^T A e)(e^T B e),$$

where $D = C + Aee^T B$ and $\mathrm{LAP}(D)$ is the linear assignment problem with cost matrix D. By construction (see [2, Section 3])

$$\mathrm{PB}(A, B, C) \leq \mathrm{QPB}(A, B, C) \leq \mathrm{QAP}(A, B, C),$$

and [2, Lemma 1] $\mathrm{PB}(A,B,C) < \mathrm{QPB}(A,B,C)$ if $\mathrm{PB}(A,B,C) < \mathrm{QAP}(A,B,C)$ and the solution of $\mathrm{LAP}(D)$ is unique.

The constraints $Xe = X^T e = e$ may be written in the form $F\,\mathbf{vec}(X) = e$, where

$$F = \begin{pmatrix} e^T \otimes I \\ I \otimes e^T \end{pmatrix}.$$

It is easy to show that the columns of $V \otimes V$ are a basis for the nullspace of F (see for example [1, Lemma 3.3]). It then follows from the definition of Q, and the fact that $\hat{S}$ and $\hat{T}$ are feasible for $\mathrm{SDD}(\hat{A},\hat{B})$, that $(V^T \otimes V^T)Q(V \otimes V) \succeq 0$, or equivalently Q is positive semidefinite on the nullspace of F. Therefore $\mathrm{QPB}(A,B,C)$ is a convex quadratic programming problem.

In [2], $\mathrm{QPB}(A,B,C)$ is computed using an interior-point algorithm. This approach allows for a high-accuracy solution, but for the dimensions of interest to us would be too time-consuming for use in a branch-and-bound context. The methodology we employ here is to approximately solve $\mathrm{QPB}(A,B,C)$ using the well-known Frank-Wolfe (FW) algorithm [15]. Although the FW algorithm is provably globally convergent, the method is known to have poor asymptotic performance. This is not of great concern to us since $\mathrm{QPB}(A,B,C)$ is only being used to obtain a bound on $\mathrm{QAP}(A,B,C)$. An excellent feature of the FW algorithm in our application is that the work on each iteration is dominated by the solution of a linear assignment problem (LAP), which can be performed extremely rapidly.

Let $f(X) = \mathbf{vec}(X)^T Q\,\mathbf{vec}(X) + C \bullet X + \langle \lambda(\hat{A}), \lambda(\hat{B}) \rangle_-$, and let

$$G = G(X) = 2(AXB - SX - XT) + C \tag{1}$$

denote the gradient of $f(\cdot)$ at X,

$$G_{ij}(X) = \frac{\partial f(X)}{\partial X_{ij}}.$$

Each iteration $k \geq 0$ of the FW algorithm begins with a feasible solution X_k for $\mathrm{QPB}(A,B,C)$. Let $G_k = G(X_k)$, and let X_k^* be an optimal solution of $\mathrm{LAP}(G_k)$. The algorithm then takes a step of the form

$$X_{k+1} = X_k + \alpha(X_k^* - X_k), \tag{2}$$

where $0 \leq \alpha \leq 1$ is chosen so as to minimize $f(X_{k+1})$. Note that since $f(\cdot)$ is quadratic the computation of the minimizing α is trivial. Substituting (2) into (1), we find that

$$G_{k+1} = (1-\alpha)G_k + \alpha[2(AX_k^*B - SX_k^* - X_k^*T) + C].$$

Since X_k^* is always a permutation matrix, the $O(n^3)$ work on an iteration of the FW algorithm can be reduced to a single matrix multiplication (to compute AX_k^*B), and the solution of one linear assignment problem.

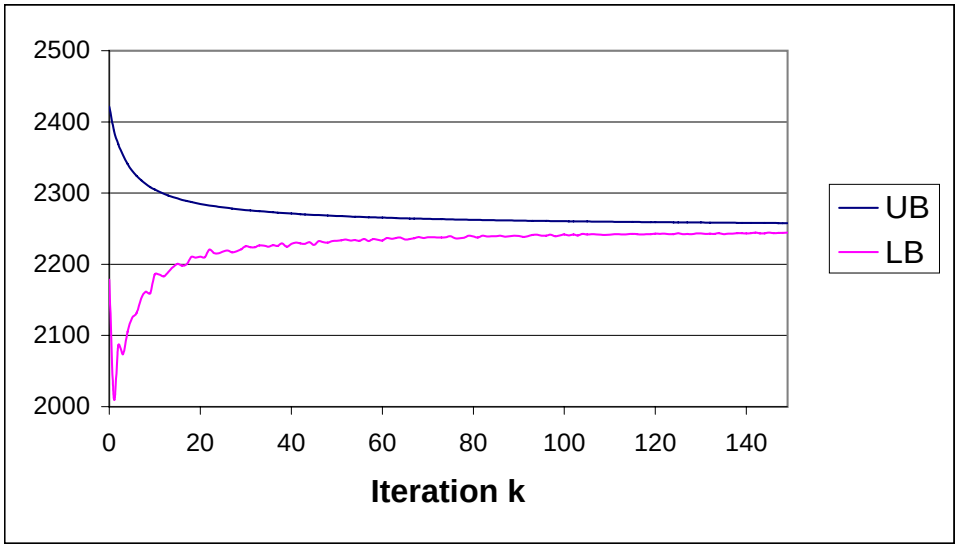


FIGURE 1: FW iterations on nug20 root problem

Associated with an optimal solution X_k^* of $\mathrm{LAP}(G_k)$ is a matrix of "reduced costs" U_k, such that for any X with $Xe = X^T e = e$,

$$G_k \bullet X = \mathrm{LAP}(G_k) + U_k \bullet X. \tag{3}$$

Since $f(\cdot)$ is convex on the nullspace of F, for any X feasible in $\mathrm{QAP}(A, B, C)$ we have

$$\begin{aligned} f(X) &\geq f(X_k) + G_k \bullet (X - X_k) && (4)\\ &= f(X_k) - G_k \bullet X_k + \mathrm{LAP}(G_k) + U_k \bullet X \\ &= f(X_k) + G_k \bullet (X_k^* - X_k) + U_k \bullet X \\ &= z_k + U_k \bullet X, && (5) \end{aligned}$$

where

$$z_k = f(X_k) + G_k(X_k^* - X_k) = f(X_k) - U_k \bullet X_k, \tag{6}$$

and the second equality in (6) follows from (3) and the fact that $U_k \bullet X_k^* = 0$. Since $U_k \bullet X \geq 0$ for any $X \geq 0$, (5) implies that on each iteration k of the FW algorithm we have a lower bound $z_k \leq \mathrm{QPB}(A, B, C) \leq \mathrm{QAP}(A, B, C)$.

In Figure 1 we illustrate the behavior of the FW algorithm in approximately solving the problem $\mathrm{QPB}(A, B, C)$ associated with the nug20 QAP. The algorithm is initialized at $X_0 = (1/n)ee^T$. In the figure we plot the sequence of upper bounds (UB) $v_k = f(X_k)$, and lower bounds (LB) z_k, $k \geq 0$. Note that the lower bounds are clearly non-monotonic (in particular $z_1 < z_0$) and the asymptotic behavior appears to be poor, as expected. On the other hand the lower bound z_k obtained after about $k = 100$ FW iterations is a reasonable approximation of $\mathrm{QPB}(A, B, C)$.

On a fast workstation z_{100} can be obtained for a problem with $n = 20$ in less than 0.1 seconds of CPU time.

For the nug20 problem the initial bound z_0 generated by the FW algorithm is 2178.3, which is precisely the value of the projected eigenvalue bound for the same problem. This is not a coincidence, as demonstrated in the following lemma.

Lemma 2.1. *Suppose that the Frank-Wolfe algorithm is applied to* $\mathrm{QPB}(A, B, C)$ *with the initial solution* $X_0 = (1/n)ee^T$. *Then* $z_0 = \mathrm{PB}(A, B, C)$.

Proof. From (1),

$$G_0 = \frac{2}{n}\left(Aee^TB - See^T - ee^TT\right) + C = \frac{2}{n}(Aee^TB) + C,$$

because $S = V\hat{S}V^T$, $T = V\hat{T}V^T$, and $e^TV = 0$. Similarly

$$\begin{aligned} f(X_0) &= \mathbf{vec}(X_0)^T Q\,\mathbf{vec}(X_0) + C \bullet X_0 + \langle\lambda(\hat{A}), \lambda(\hat{B})\rangle_- \\ &= \mathrm{tr}(AX_0BX_0^T - SX_0X_0^T - X_0TX_0^T) + C \bullet X_0 + \langle\lambda(\hat{A}), \lambda(\hat{B})\rangle_- \\ &= \frac{1}{n^2}\mathrm{tr}(Aee^TBee^T) + C \bullet X_0 + \langle\lambda(\hat{A}), \lambda(\hat{B})\rangle_- \\ &= \frac{1}{n^2}(e^TAe)(e^TBe) + C \bullet X_0 + \langle\lambda(\hat{A}), \lambda(\hat{B})\rangle_-. \end{aligned}$$

Let $D = G_0 = C + (2/n)Aee^TB$. From (6) we then have

$$\begin{aligned} z_0 &= f(X_0) + \mathrm{LAP}(D) - G_0 \bullet X_0 \\ &= \langle\lambda(\hat{A}), \lambda(\hat{B})\rangle_- + \mathrm{LAP}(D) + \frac{1}{n^2}(e^TAe)(e^TBe) - \frac{2}{n}(Aee^TB) \bullet X_0 \\ &= \langle\lambda(\hat{A}), \lambda(\hat{B})\rangle_- + \mathrm{LAP}(D) - \frac{1}{n^2}(e^TAe)(e^TBe) \\ &= \mathrm{PB}(A, B, C). \end{aligned}$$

□

The behavior illustrated in Figure 1 is typical in our experience. When initialized at $X_0 = (1/n)ee^T$, the bound z_1 drops sharply from the inital value $z_0 = \mathrm{PB}(A, B, C)$, and the bound sequence z_k, $k \geq 1$ then increases relatively steadily. We have experimented with alternative initializations for the FW algorithm, and schemes for enforcing monotonicity on the bound sequence, but none of these efforts have produced reliable improvements in the overall performance of the algorithm.

In practice we make a small modification of the FW procedure described above based on the use of a matrix $\tilde{G}_k \leq G_k$. Since $X \geq 0$ for any feasible X, it follows from (4) that

$$\begin{aligned} f(X) &\geq f(X_k) + \tilde{G}_k \bullet X - G_k \bullet X_k \\ &= f(X_k) - G_k \bullet X_k + \mathrm{LAP}(\tilde{G}_k) + \tilde{U}_k \bullet X \\ &= \tilde{z}_k + \tilde{U}_k \bullet X, \end{aligned}$$

where $\tilde{U}_k \geq 0$ is the matrix of reduced costs from $\mathrm{LAP}(\tilde{G}_k)$,

$$\tilde{z}_k = \langle \lambda(\hat{A}), \lambda(\hat{B}) \rangle_- + f(X_k) - G_k \bullet X_k + \tilde{G}_k \bullet \tilde{X}_k ,$$

and $\tilde{X}_k$ is an optimal solution of $\mathrm{LAP}(\tilde{G}_k)$. By using a matrix $\tilde{G}_k$ in place of G_k we can restrict the LAP solved on each FW iteration to have integer data. In particular, if

$$\tilde{G}_k = \frac{1}{\theta_k} \lfloor \theta_k G_k \rfloor ,$$

where θ_k is a positive scaling factor, then $\tilde{G}_k \leq G_k$, $\mathrm{LAP}(\tilde{G}_k) = (1/\theta_k)\,\mathrm{LAP}(\theta_k \tilde{G}_k)$, and $\theta_k \tilde{G}_k$ is an integer matrix. By using a larger value of θ_k, $\mathrm{LAP}(\tilde{G}_k)$ becomes a better approximation of $\mathrm{LAP}(G_k)$, but the time to solve $\mathrm{LAP}(\theta \tilde{G}_k)$ typically increases. (It is well known that the time to solve a LAP with integer data is sensitive to the scale of the data.) In addition to providing a speed improvement, the use of LAPs with integer data prevents roundoff error and improves the robustness of our algorithm. In our implementation each integer LAP is solved using the well-known augmenting path algorithm of [23], chosen for its excellent performance on small dense LAPs (see [23]) as encountered in our application. The spectral decompositions required for the construction of QPB are performed using routines from the Meschach library [33].

3 BRANCHING STRATEGY

In our branch-and-bound implementation we employ "polytomic" branching, as introduced for QAP in [29] and used in numerous subsequent implementations. At a given node there is a set of fixed assignments $X_{i,\pi(i)} = 1$, $i \in \bar{I}$. Let $N = \{1, 2, \ldots, n\}$, $\bar{J} = \{\pi(i) \mid i \in \bar{I}\}$, $I = N \setminus \bar{I}$, $J = N \setminus \bar{J}$. If the node is not fathomed, we generate children according to one of the following schemes:

Row Branching. Fix $i \in I$. Generate a child problem for each $j \in J$ for which the problem with $X_{ij} = 1$ cannot be eliminated.

Column Branching. Fix $j \in J$. Generate a child problem for each $i \in I$ for which the problem with $X_{ij} = 1$ cannot be eliminated.

In both of the above cases the elimination of children is based on the dual matrix U or $\tilde{U}$ described in the previous section. For simplicity we assume throughout this section that $\mathrm{LAP}(G_k)$, rather than $\mathrm{LAP}(\tilde{G}_k)$, is solved on each iteration k of the FW algorithm (see the previous section for the distinction). The modifications required when $\tilde{G}_k$ is used are straightforward. Consider the root node, and let v denote the incumbent objective value; that is, the objective value for the best known feasible solution of $\mathrm{QAP}(A, B, C)$. Let $z = z_k$ denote the lower bound (6) produced after k FW iterations, and let $U = U_k$ be the corresponding reduced cost matrix. Since $U \geq 0$ and $X \geq 0$, (5) implies that if $z + u_{ij} \geq v$, then no solution with $X_{ij} = 1$ can have objective value less than v, and therefore this potential child can be eliminated. Using U in this way is similar to the use of the matrix of

reduced costs associated with the "master" LAP for the GLB to eliminate potential children, as in [29].

We next describe several different rules that we employ in choosing the row i or column j on which to branch. For simplicity we first describe all the rules in terms of row branching. Column branching, and the treatment of problem symmetries, are described later. The rules are specified as they would be implemented at the root node ($\bar{I} = \bar{J} = \emptyset$). At an arbitrary node in the tree the problem is of the form QAP(A', B', C') where the matrices A', B', C' are all of dimension $n - |\bar{I}|$, and the implementation of branching rules is similar.

Our first two branching rules use the information in U to try to increase the child bounds as much as possible, or alternatively minimize the number of children created.

Rule 1. Branch on the row i that has the highest value of $\sum_{j \in N} U_{ij}$.

Rule 2. Branch on the row i that produces the smallest number of children. In the event of a tie, choose the row with the largest value $\sum_{j \in N_i'} U_{ij}$, where $N_i' = \{j \in N \mid z + u_{ij} < v\}$.

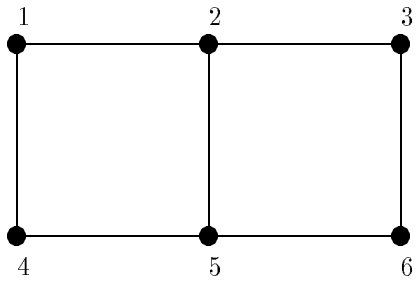
Note that the set N_i' in Rule 2 consists exactly of the child problems with $X_{ij} = 1$ that *cannot* be eliminated. Rule 2 is a straightforward analog of the branching rule used in [29], and is effective in reducing the size of the tree for small problems. We make extensive use of Rule 2 once we are deep enough in the tree so that a non-trivial fraction of the children can be eliminated. At lower depths in the tree on large problems, however, it is typically the case that no children can be eliminated. In this case we also consider branching rules that obtain more information by "prospectively" setting $X_{ij} = 1$, and computing QPB for the associated QAP problem of dimension $n - 1$, before making the final decision of where to branch. This is analogous to the well-known technique of "strong branching" for integer and mixed-integer linear programming, see for example [27].

Rule 3. Let I_1 denote the set of rows having the k_1 highest values of $\sum_{j \in N} U_{ij}$. For each $i \in I_1$, and $j \in N$, compute a lower bound $z^{ij} = z^{ij}_{k_2}$ by forming the reduced problem QAP(A', B', C') corresponding to $X_{ij} = 1$, and approximately solving QPB(A', B', C') using k_2 FW iterations. Branch on the row $i \in I_1$ having the highest value of $\sum_{j \in N} z^{ij}$.

In Rule 3 the z^{ij} values are only computed for a subset of the k_1 most promising rows, based on Rule 1. The purpose of this restriction is to economize the time required to compute the z^{ij}. In addition, the number of FW iterations k_2 used to compute the z^{ij} bounds is typically less than the number of iterations used to compute the lower bound z at the current node, again to reduce the computation time.

Our final rule is an elaboration of Rule 3. Note that when a prospective bound z^{ij} is computed in the course of Rule 3, there is an associated dual matrix U^{ij}, of dimension $n - 1$. Rule 4 is based on applying Rule 1 to each of these matrices

FIGURE 2: Grid for distances in nug06 QAP



U^{ij}. Rule 4 can be thought of as a "look-ahead" branching rule, where we try to maximize the total increase in the bounds after 2 levels of branching.

Rule 4. Let I_1 denote the set of rows having the k_1 highest values of $\sum_{j \in N} U_{ij}$. For each $i \in I_1$, and $j \in N$, compute a lower bound $z^{ij} = z^{ij}_{k_2}$ by forming the reduced problem QAP(A', B', C') corresponding to $X_{ij} = 1$, and approximately solving QPB(A', B', C') using k_2 FW iterations. Let U^{ij} be the reduced cost matrix associated with z^{ij}. Let v^{ij} be the maximal row sum of U^{ij}, and let $w^{ij} = (|N| - 1)z^{ij} + v^{ij}$. Branch on the row $i \in I_1$ having the highest value of $\sum_{j \in N} w^{ij}$.

In practice we implement Rules 1-4 in a somewhat more complex fashion than described above. In particular, we consider column branching as well as row branching, and symmetries in the problem data. Below we describe the details associated with these extensions.

Symmetry. Many QAP problems arise from applications where the distance matrix B exhibits symmetries that can be exploited to reduce the number of child problems generated in the branching process. Logic for exploiting such symmetries was introduced in [29], and has been used in many subsequent branch-and-bound implementations. In a problem with symmetries there is a subset of the locations J_1 such that without loss of generality the children of the root problem can be restricted to be of the form $X_{ij} = 1$, $j \in J_1$, regardless of the row i. In addition, there may be one or more pairs of subsets of locations $\{J_2, J_3\}$ so that if at any node in the tree the set of fixed locations $\bar{J}$ satisfies $\bar{J} \subset J_2$, then the children can be restricted to be of the form $X_{ij} = 1$, $j \in J_3$, regardless of the choice of $i \in I$. For a simple example consider the problem nug06. The distance matrix for this problem corresponds to the l_1 distances on a 2×3 rectangular grid, as shown in Figure 2. In this case we may take $J_1 = \{1, 2\}$, $J_2 = \{2, 5\}$, $J_3 = \{1, 2, 4, 5\}$. Larger problems may have more than one pair of $\{J_2, J_3\}$ subsets. At a node where symmetry can be exploited we consider only row branching, and replace the index set N used in the branching rules with an appropriate $J \subset N$.

Column Branching. When symmetry is *not* present at a node we consider column branching as well as row branching. The modifications to the branching rules are straightforward. For Rule 1 we choose the row i with the highest row sum from

U, or the column j with the highest column sum, whichever is higher. For Rule 2 we branch on the row or column which produces the smallest number of children. For Rule 3 we choose the k_1 most promising rows *and* k_1 most promising columns, compute all the required z^{ij} bounds, and then choose the row or column with the highest sum. Rule 4 is similar, with v^{ij} corresponding to the maximal row or column sum of U^{ij}.

To completely specify the above branching rules a number of parameters must be chosen. These are:

NFW1. Maximum number of FW iterations used.

NFW2. Maximum number of FW iterations used if node cannot be fathomed.

NFW3. Number of FW iterations used for prospective bound computations (Rules 3 and 4 only; same as k_2 in description above).

NBEST. Number of rows/columns in which to compute prospective bounds (Rules 3 and 4 only; same as k_1 in description above).

UPDATE. Number of FW iterations between update of dual matrices S, T.

We now give some details regarding these parameters. On each FW iteration we obtain an objective value $v_k = f(X_k)$ and lower bound z_k. If $z_k > v$ the current node can be fathomed, and the FW process is terminated. On the other hand if $v_k < v$ then we know that the lower bound from QPB will *not* be high enough to fathom the current node. By setting NFW2<NFW1 we allow for earlier termination in the latter case. (Note however that even when a node cannot be fathomed it is desirable to compute a reasonably accurate bound z for branching purposes.) The complete logic for the number of FW iterations is then that we terminate on iteration k if $z_k > v$, or $v^k < v$ and $k \geq$ NFW2, or k=NFW1.

As described in the previous section, QPB(A, B, C) involves the choice of matrices $\hat{S}$, $\hat{T}$ that are optimal in SDD$(\hat{A}, \hat{B})$. Such an optimal solution is not in general unique, and different choices of $\hat{S}$, $\hat{T}$ may produce different values of QPB(A, B, C). Given a choice of $\hat{S}$, $\hat{T}$, and an approximate solution X of QPB(A, B, C), a simple procedure is described in [2, Section 6] that attempts to generate a new dual optimal solution that will increase QPB(A, B, C). We apply one step of this procedure every UPDATE FW iterations, using the current primal solution X_k as the basis for the update.

In general we combine the 4 different branching rules given above, with suitable parameter choices, to obtain a complete branching strategy. In the implementation described here the choice of branching rule is based on depth in the tree, that is, the number of fixed assignments $|\bar{I}|$, and the tree is traversed using depth-first search [27]. Our simplest branching strategy, Strategy A, is shown in Table 1. Strategy A uses Rule 2, with NFW1=150, NFW2=100, updating the dual matrices every 30 FW iterations, for all levels in the tree (note that with polytomic branching the maximum level in the tree is n). A more complex branching strategy, Strategy B, is given in Table 2. Strategy B uses Rule 4 on levels 0 and 1 of the tree, Rule 3 on level 2, and Rule 2 at all higher levels. In the next section we consider computational results obtained using Strategies A and B, and two more strategies C and D that

TABLE 1: Branching strategy A

Depth	Rule	NFW1	NFW2	NFW3	NBEST	UPDATE
50	2	150	100	–	–	30

TABLE 2: Branching strategy B

Depth	Rule	NFW1	NFW2	NFW3	NBEST	UPDATE
1	4	150	100	50	20	30
2	3	100	100	25	10	30
50	2	75	50	–	–	30

are simple modifications of Strategy B. In Strategy C the "cutoffs" for Rules 4 and 3 are set at levels 1 and 3, respectively, and in Strategy D the cutoffs are set at levels 2 and 4, respectively. In general the use of Rules 3 and 4 deeper in the tree is appropriate for larger, more difficult problems.

4 COMPUTATIONAL RESULTS

In this section we describe the performance of our B&B algorithm on a set of QAP test problems from QAPLIB [9]. We begin with a small example that illustrates the effect of different branching strategies. We consider the problem scr15, and implement our B&B algorithm using branching strategies A and B, as described in the previous section. The results are given in Table 3. In the table the entries in the "Fthm." column give the fraction of the nodes at each level that were fathomed. For the remaining unfathomed nodes, the entries in the "Elim." column give the fraction of the potential child nodes that were eliminated. For example, using strategy B, 23% of the nodes on level 3 were fathomed, and 70% of the potential children of the unfathomed nodes were eliminated. Note that for both strategies the number of level 1 nodes is 9, rather than 15, due to symmetry in the distance matrix for the problem. Using rules 4 and 3 for the top few levels of the tree, in Strategy B, incurs extra time at those levels compared to Strategy A, but pays off handsomely in greatly reducing the size of the tree. Note that the total number of nodes required using Strategy A is higher by a factor of over 80, and the total time is higher by a factor of about 13. All times in the table are CPU seconds on an HP9000 C3000 workstation. In our implementation QAP subproblems of size $n = 3$ are solved by enumeration; for scr15 this occurs on level 12.

Below we give computational results that compare the performance of our algorithm with several other recent B&B algorithms for the QAP. We consider a number of "medium to large" ($16 \leq n \leq 24$) problems from QAPLIB [9] that are commonly

TABLE 3: Comparison of branching strategies on scr15

	Branching Strategy A				Branching Strategy B			
Level	Nodes	Fthm.	Elim.	Time	Nodes	Fthm.	Elim.	Time
0	1	0.00	0.00	0.05	1	0.00	0.00	1.36
1	9	0.00	0.00	0.34	9	0.00	0.17	11.43
2	108	0.06	0.09	2.96	90	0.07	0.80	44.50
3	1,159	0.24	0.16	21.57	213	0.23	0.70	2.45
4	8,900	0.50	0.27	97.17	601	0.68	0.55	3.33
5	35,218	0.68	0.38	228.25	949	0.80	0.67	2.90
6	69,775	0.80	0.49	283.43	615	0.88	0.67	1.35
7	64,436	0.87	0.52	165.40	217	0.97	0.75	0.31
8	31,418	0.92	0.50	53.82	14	0.93	0.86	0.00
9	8,631	0.98	0.56	7.75	1	0.00	0.83	0.00
10	538	1.00	0.70	0.26	1	0.00	0.80	0.00
11	3	0.67	0.75	0.00	1	0.00	0.75	0.00
12	1			0.00	1			0.00
Total	220,197			861.00	2,713			67.63

used as benchmarks. In applying our algorithm we choose one of the branching strategies (A, B, C, or D) described in the previous section, based on the size and difficulty of the problem. In Table 4 we give the number of nodes required in the B&B tree for the optimal solution of each problem using our algorithm (BA). In each case the initial incumbent value v is set to the optimal objective value of the problem plus one. (For many of these problems the optimal objective value was first proved optimal within the last 5 years.) With this incumbent value the algorithm cannot fathom any node on a path that leads to an optimal solution, so the B&B algorithm must complete one "dive" in the tree that leads to an optimal permutation. For comparison we give the number of nodes required by the B&B algorithms of Clausen and Perregaard (CP) [12], Brüngger et al. (BMCP) [6], Hahn, Grant and Hall (HGH) [19], and Hahn et al. (HHJGR) [20]. The results from CP and BMCP use the same GLB-based algorithm; the results for nug16b/18/20 are from CP, and for the remainder of the problems are from BMCP. HGH and HHJGR both use the dual-LP bound of [18] (see also [25]), but HHJGR uses more sophisticated branching rules. The HHJGR results for had16 and nug18 were provided by Hahn (private communication).

From Table 4 it is clear that in terms of nodes our results are far superior to the GLB-based results of CP/BMCP, and are of the same order of magnitude as the node counts from HGH/HHJGR on all problems except rou20. However, since GLB can be computed much more rapidly than either our QP-based bound, or the dual-LP bound used by HGH/HHJGR, a consideration of the total CPU time required for the different methods is required. Because of advances in hardware, and the use

TABLE 4: Comparison of nodes for B&B algorithms on QAPLIB problems

Problem	Strategy	BA	CP/BMCP	HGH	HHJGR
had16	A	8,964	18,770,885	13,808	3,069
had18	A	104,229	761,452,218	197,487	53,224
had20	A	122,460	7,616,968,110		
nug16b	B	6,867	320,556		
nug18	C	251,470	114,948,381		202,021
nug20	C	1,040,308	360,148,026	724,289	239,449
nug21	C	1,698,093	3,631,929,368	3,192,565	
nug22	C	1,225,892	48,538,844,41	10,768,366	988,302
nug24	D	31,865,440			11,674,95
rou20	D	56,082,781	2,161,665,137	2,090,862	
tai17a	C	750,441	20,863,039		
tai20a	D	54,643,195	2,215,221,637		

of parallel machines for some references, raw CPU times must be adjusted with care to make a sensible comparison. Our times are obtained on a single, lightly loaded HP9000 C3000 workstation. In comparing our times to the results of CP/BMCP and HGH/HHJGR the raw times reported in these papers are adjusted as follows.

HHJGR. Times in this paper were obtained on a 360 MHz Sun UltraSparc workstation. Based on the SPECint95 figure of 15.2 for this machine, versus 31.8 for the HP9000 C3000, the times reported in [20] were divided by 2.

HGH. Times in this paper were obtained on a 75 MHz Sun SuperSparc. Based on information provided by Hahn (private communication), the UltraSparc used by HHJGR is approximately 4 times as fast as this machine on the QAP B&B application. Consequently the times reported in [19] were divided by 8.

CP. Times in this paper were obtained on a 16-processor MEIKO computing system. It is reported in [11] that the 16-processor system provides an effective "speed-up" of about 14 over the use of one processor, and a single processor in the MEIKO system requires 3-4 times as much time as an HP9000/735 workstation on the same application. Using a factor of 3.5 to convert to the HP9000/735, the raw times in [12] times 4 would give approximate run times on a single HP9000/735. This HP machine appears to be comparable in performance to the Sun SuperSparc used in [19] (SPECint95 figures are 3.27 for the HP and 3.11 for the Sun). Thus the raw times in [12] are approximately equal to the serial time on the Sun UltraSparc used by HHJGR, and these times were divided by two to convert to the HP9000 C3000.

BMCP. Times in this paper were obtained on NEC Cenju-3 and Intel Paragon XP/S22 multi-processor systems. The only information available for the performance of individual processors in these systems are MFLOP rates of 50 for

TABLE 5: Equivalent CPU time (m) on QAPLIB problems

Problem	BA	CP/BMCP	HGH	HHJGR
had16	0.8	15.8	2.8	0.4
had18	11.1	871.1	70.1	9.1
had20	15.7	50,992.4		
nug16b	0.9	0.3		
nug18	69.2	123.1		33.5
nug20	145.8	483.0	333.1	134.1
nug21	212.3	6,331.9	1,296.9	
nug22	134.3	113,336.1	3,775.9	1,663.4
nug24	5829.9			6,437.7
rou20	8391.6	3,787.7	2,062.6	
tai17a	112.7	23.6		
tai20a	5505.2	12,131.8		

the NEC, and 75 for the Intel. The number of processors used to solve different problems varied between 16 and 96. Times were converted by assuming a parallel utilization factor of .875 (as also used in [19]), converting to serial time on the Sun SuperSparc based on that machine's MFLOP rate of 44.4, and finally dividing by 8 to convert to the HP9000 C3000.

In Table 5 we give the CPU time (in minutes) required by our algorithm, and equivalent times for the B&B algorithms of CP/BMCP, HGH, and HHJGR on the same problems, obtained as described above. Clearly the adjustment is most difficult for the BMCP times, but even there the adjusted times appear to be quite reasonable. Note, for example, that the ratio of the nodes required for nug21 (from BMCP) compared to nug20 (from CP) is 10.1, compared to a ratio of equivalent times of 13.1. It is to be expected that the time ratio would be somewhat greater than the node ratio, because the problems at each level of the tree are larger.

Table 5 indicates that the performance of our algorithm on most larger problems is far superior to the GLB-based results of CP/BMCP, and is competitive with the state-of-the-art dual-LP based results of HHJGR. It should be noted that the results reported here for our algorithm do *not* use highly optimized branching strategies. In particular, by using more complex problem-specific branching strategies we can reduce both the nodes and time required by our algorithm. However our results indicate that even with simple branching strategies the performance of our algorithm on difficult problems is highly competitive.

In Figure 3 we illustrate the equivalent times from Table 5 for the nugxx problems of size 18 to 24, using a logarithmic time scale. Exponential growth is evident for all four series, but the rate of growth for our results appears to be lower than for the other algorithms. (We believe that the trend for our times is best represented if our result for nug22, which is exceptionally good, is ignored.) This observation

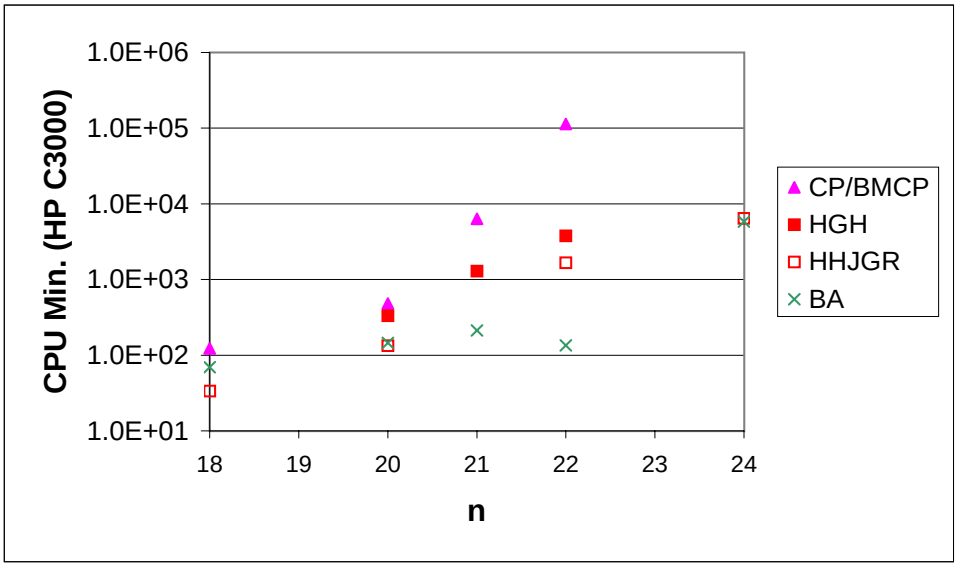


FIGURE 3: Equivalent times to solve nugxx problems

suggests that our algorithm may perform well on problems even larger than those considered here, particularly if more complex branching rules are employed. In order to obtain reasonable solution times for problems of size $n > 24$ we have implemented our algorithm using the Master-Worker (MW) distributed processing system [16]. Details of the MW implementation and computational results on larger problems are reported in [3].

REFERENCES

1. K.M. Anstreicher (2001). Eigenvalue bounds versus semidefinite programming for the quadratic assignment problem. *SIAM J. Optimization*, **11**, 254-265.
2. K.M. Anstreicher and N.W. Brixius (2001). A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming*, **89**, 341-357.
3. K.M. Anstreicher, N.W. Brixius, J.-P. Goux, and J. Linderoth (2001). Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, to appear.
4. K.M. Anstreicher and H. Wolkowicz (2000). On Lagrangian relaxation of quadratic matrix constraints. *SIAM J. Matrix Analysis and Applications*, **22**, 41-55.
5. M.S. Bazaraa and O. Kirca (1983). A branch-and-bound based heuristic for solving the quadratic assignment problem. *Naval Research Quarterly*, **30**, 287-304.
6. A. Brüngger, A. Marzetta, J. Clausen, and M. Perregaard (1998). Solving large-scale QAP problems in parallel with the search library ZRAM. *Journal of Parallel and Distributed Computing*, **50**, 157-169.
7. R.E. Burkard, E. Çela, P.M. Pardalos, and L.S. Pitsoulis (1998). The quadratic assignment problem. In D.-Z. Zhu and P.M. Pardalos, Editors, *Handbook of Combinatorial Optimization*, Vol. 3, Kluwer.
8. R. Burkard and U. Derigs (1980). *Assignment and Matching Problems: Solution Methods with FORTRAN Programs*. Springer, Berlin.

9. R.E. Burkard, S. Karisch, and F. Rendl (1997). QAPLIB - A quadratic assignment problem library. *J. Global Optimization*, **10**, 391-403.
10. E. Çela (1998). *The Quadratic Assignment Problem: Theory and Algorithms.* Kluwer.
11. J. Clausen, S.E. Karisch, M. Perregaard, and F. Rendl (1998). On the applicability of lower bounds for solving rectilinear quadratic assignment problems in parallel. *Computational Optimization and Applications*, **10**, 127-147.
12. J. Clausen and M. Perregaard (1997). Solving large quadratic assignment problems in parallel. *Computational Optimization and Applications*, **8**, 111-128.
13. J. Crouse and P. Pardalos (1989). A parallel algorithm for the quadratic assignment problem. *Proceedings of Supercomputing '89*, ACM Press, 351-360.
14. G. Finke, R.E. Burkard, and F. Rendl (1987). Quadratic assignment problems. *Annals of Discrete Mathematics*, **31**, 61-82.
15. M. Frank and P. Wolfe (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, **3**, 95-110.
16. J.-P. Goux, J. Linderoth, and M. Yoder (2000). Metacomputing and the master-worker paradigm. Preprint ANL/MCS-P792-0200, MCS Division, Argonne National Laboratories, Chicago, IL.
17. A. Graham (1981). *Kronecker Products and Matrix Calculus: with Applications.* Ellis Horwood, Chichester.
18. P. Hahn and T. Grant (1998). Lower bounds for the quadratic assignment problem based upon a dual formulation. *Operations Research*, **46**, 912-922.
19. P. Hahn, T. Grant and N. Hall (1998). A branch-and-bound algorithm for the quadratic assignment problem based on the Hungarian method. *European Journal of Operational Research*, **108**, 629-640.
20. P.M. Hahn, W.L. Hightower, T.A. Johnson, M. Gugnard-Spielberg, and C. Roucairol (1999). Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem. Dept. of Systems Engineering, University of Pennsylvania, Philadelphia PA.
21. S.W. Hadley, F. Rendl, and H. Wolkowicz (1992). A new lower bound via projection for the quadratic assignment problem. *Mathematics of Operations Research*, **17**, 727-739.
22. R.A. Horn and C.R. Johnson (1991). *Topics in Matrix Analysis.* Cambridge University Press, Cambridge.
23. R. Jonker and A. Volgenant (1987). A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, **38**, 325-340.
24. V. Kaibel (2000). Polyhedral Methods for the QAP. In P.M. Pardalos and L. Pitsoulis, Editors, *Nonlinear Assignment Problems*, Kluwer.
25. S.E. Karisch, E. ÇELA, J. Clausen, and T. Espersen (1999). A dual framework for lower bounds of the quadratic assignment problem based on linearization. *Computing*, **63**, 351-403.
26. S.E. Karisch and F. Rendl (1995). Lower bounds for the quadratic assignment problem via triangle decompositions. *Mathematical Programming*, **71**, 137-152.
27. J.T. Linderoth and M.W.P. Savelsbergh (1999). A computational study of search strategies in mixed integer programming. *INFORMS J. Computing*, **11**, 173-187.
28. A. Marzetta and A. Brüngger (1999). A dynamic-programming bound for the quadratic assignment problem. In T. Assano et al., Editors, *COCOON'99, Lecture Notes in Computer Science*, **1627**, pp. 339-348, Springer, Berlin.
29. T. Mautor and C. Roucairol (1994). A new exact algorithm for the solution of quadratic assignment problems. *Discrete Applied Mathematics*, **55**, 281-293.
30. M.W. Padberg and M.P. Rijal (1996). *Location, Scheduling, Design and Integer Programming*, Kluwer.
31. P. Pardalos, F. Rendl, and H. Wolkowicz (1994). The quadratic assignment problem: A survey and recent developments. In *Quadratic Assignment and Related Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, **16**, pp. 1-41, American Mathematical Society.

32. P.M. Pardalos, K.G. Ramakrishnan, M.G.C. Resende, and Y. Li (1997). Implementation of a variance reduction-based lower bound in a branch-and-bound algorithm for the quadratic assignment problem. *SIAM J. Optimization*, **7**, 280-294.
33. D.E. Stewart and Z. Leyk (1994). Meschach: Matrix computations in C. *Proceedings of the Center for Mathematics and its Applications*, **32**, The Australian National University.