

A Branch-and-Bound Algorithm for the Quadratic Assignment Problem Based on the
Hungarian Method

by Peter Hahn

Sci-Tech Services

1416 Park Rd., Elverson, PA 19520,

Thomas Grant

Bear Stearns and Co.

245 Park Ave., NY, NY 10167,

and Nat Hall

Flexible Intelligent Technology

P.O.B. 5454, Herndon, VA 22070

Abstract

This paper presents a new branch-and-bound algorithm for solving the Quadratic Assignment Problem (QAP). The algorithm is based on a Dual Procedure (DP) similar to the Hungarian method for solving the Linear Assignment Problem. Our DP solves the QAP in certain cases, i.e., for some small problems ($N < 7$) and for numerous larger problems ($7 \leq N \leq 16$) that arise as sub-problems of a larger QAP such as the Nugent 20. The DP, however, does not guarantee a solution. It is used in our algorithm to calculate lower bounds on solutions to the QAP. As a result of a number of recently developed improvements, the DP produces lower bounds that are as tight as any which might be useful in a branch-and-bound algorithm. These are produced relatively cheaply, especially on larger problems. Experimental results show that the computational complexity of our algorithm is lower than known methods, and that its actual runtime is significantly shorter than the best known algorithms for QAPLIB test instances of size 16 through 22. Our method has the potential for being improved and therefore can be expected to aid in solving even larger problems.

Keywords

Quadratic Assignment Problem,

Branch-and-bound,

Quadratic Programming,

Integer Programming,

Mathematical Programming.

1. Introduction

This paper describes a branch-and-bound algorithm for solving the Quadratic Assignment Problem (QAP). The algorithm uses for its bounding method a mathematical dual of the problem, which has roots in the Hungarian algorithm (see Munkres [21]) which solves the linear assignment problem (LAP). This bounding method, the Dual Procedure (DP), was first discovered by one of us (Hahn) [15] in an attempt to solve the QAP via a resolvent sequence approach originally formulated by House, et al [17]. The work has remained unpublished since 1968. Having at first pursued the resolvent sequence approach without great success, we turned to developing the DP for calculating lower bounds. As a recent step, we devised a branch-and-bound algorithm incorporating the DP in order to solve the QAP exactly. Continuous improvements in our implementation have permitted us to achieve solution run times which are at the leading edge for single processor platforms.

The DP bears strong resemblance to other existing lower bounding techniques, such as Assad and Xu [2], Burkard [6], Carraresi and Malucelli [8], Gilmore [12], Lawler [19], and Roucairol [25]. The difference lies in the DP's broad class of operations which permit more powerful and more computationally efficient strategies than do these others. These strategies are described in Hahn and Grant [16].

1.1 Background

The QAP covers a broad class of problems which involve the minimization of a total pair-wise interaction cost among N facilities. These problems include finding the assignment of factories to fixed locations which minimizes transportation cost and the location of sub-assemblies on a chassis in order to minimize the length of interconnecting wiring. The Koopmans-Beckman [18] version of the problem can be stated with reference to a practical situation where it is desired to locate N facilities among N fixed locations, where for each pair of facilities (i,k) a certain flow of commodities $f(i,k)$ is known and for each pair of locations (j,n) a corresponding distance $d(j,n)$ is known. The two-way transportation costs between facilities i and k , given that i is assigned to location j and k is assigned to location n , are $f(i,k) \cdot d(j,n) + f(k,i) \cdot d(n,j)$. The objective is to find an assignment minimizing the sum of all such transportation costs. In the general case of the QAP, the cost of transportation

between facilities is known but is not decomposable into a product of a flow and a distance matrix.

The QAP has long been of interest to researchers in various fields, both because of its wide applicability and also its resistance to reliable computer solution. QAP belongs to the class of NP-complete problems and is considered one of the most difficult. The importance of the classification "NP-complete" is that the run time required to solve (optimally) problems in this class cannot be bounded (from above) by a polynomial that is a function of the problem size. Rather, run time is bounded from above by a function that increases exponentially with problem size, so that it may not be possible to obtain optimal solutions for large problems. Sahni and Teogilo [27] proved that the Traveling Salesman Problem (TSP) is NP-complete. As the TSP is a special case of the QAP, the QAP is NP-complete as well. Exact solution strategies for the QAP have been largely unsuccessful for any but small problems (approximately $N \leq 20$). As a result, a significant amount of effort has been put forth by researchers in developing "inexact", or "heuristic", methods, which obtain good suboptimal assignments using a reasonable amount of CPU time (see Burkard [4] and Li [20]).

1.2 Exact Solutions

There are many cases where an exact solution is essential; when, for instance, large expenditures of time or money are involved or when suboptimal assignments need to be evaluated. Specifically, the location of large plants and storage facilities may be done only once in the lifetime of an organization but the cost of ongoing transportation between these facilities can be a determining factor in its ultimate profitability.

Laying out circuits on a VLSI chip is similarly a one-time task. Applying the QAP here in order to minimize the die size has large economic impact, because of the large number of chips of a given type produced. An additional advantage of microcircuit size reduction is the lower power consumption and increased speeds that can be achieved when run length is minimized -- resulting in reduced circuit capacitance and shorter propagation lengths. Microcircuits have tens of thousands of individual components, so that global optimization of their layout is not feasible. However, such complex problems are often broken down into a hierarchy of smaller problems. Some of these smaller problems are better solved exactly, while some could be solved with

heuristic methods. The combining of various methods has been suggested as the only way to deal with such practical and difficult problems (see Areibi and Vannelli [1]).

The majority (and most successful) of exact solution procedures for the QAP are enumerative (i.e., branch-and-bound) strategies which differ essentially in the partial assignment strategies and in the computation of lower bounds. Until now, the best lower bound functions for the Koopmans-Beckman version of the problem have been the 30 year old Gilmore-Lawler bound (see [12] and [19]). Clausen and Perregaard have chosen to use this lower bound in their ground-breaking solution of the Nugent 20 problem. Their success depended partly on improvements in the Gilmore-Lawler bounding technique and partly on improvements in the algorithm they used. The latter came from making as much use as possible of additional information generated at each partial assignment decision.

The single, most important and widely studied ingredient of branch-and-bound algorithms is the lower bounds used for fathoming partial assignments. Two aspects of these lower bounds are critical to their success. The first is their tightness, i.e., their ability to fathom partial assignments when only few partial assignments have been made. The second is their simplicity, which permits them to be computed rapidly on single processor machines. In both regards, the methods described in this paper are major improvements over what has up to now been achieved.

A final point is made regarding exact solutions. Test instances to date have been constructed artificially or produced by a random number generator. As a consequence, solution cost distributions are somewhat flat, allowing non-exact solutions to appear attractive. Real problems are likely to be much more uneven and optima may be further away than anticipated. Under such circumstances, the use of various pre-processing techniques to reduce the search space may be applicable. Nevertheless, exact solutions play an important role.

1.3 The Dual Procedure for Branch-and-bound.

The DP has a number of valuable properties that makes it ideal for use in a branch-and-bound algorithm:

- The DP is the fastest among competing lower bounding procedures as demonstrated from experimental results. (See Hahn and Grant [16].)

- Similar to a number of other lower bounding procedures, the DP generates a series of non-decreasing lower bounds on the QAP. More importantly, with each such lower bound, it generates a fully equivalent QAP. If carried to completion, the sequence of equivalent QAPs ends in the equivalent QAP which solves the dual problem. (See Section 3.2.)
- Each lower bounding calculation at a node of the tree (i.e., for a given partial assignment) is an attempt to solve a reduced size QAP (i.e., a sub-problem of the original) from a dual perspective. In many cases, the DP solves sub-problems that are only a few sizes smaller than the QAP being tackled. Thus, feasible solutions to the original QAP are generated without having to fathom deep into the search space. Consequently, when lower bounding takes place on the node (i.e., partial assignment) that contains the QAP optimum assignment, the optimum assignment is generated well before reaching an outer node (i.e., complete assignment). (See the end of Section 5.3.)
- The calculation of lower bounds is an iterative process which permits stopping early. The lower bound calculation for a given partial assignment can be stopped as soon as the lower bound on the assumed partial assignment exceeds an upper bound on the original problem. It can also be stopped, in favor of making an additional partial assignment, when it becomes obvious that from its slow progress it is unlikely to ever reach the upper bound. This is a very effective way to reduce branch-and-bound run time. (See Section 4.1.)
- In a branch-and-bound algorithm, fathoming decisions can easily be recorded by arbitrarily increasing the costs of those elements in the cost matrix that correspond to partial assignments which have been eliminated as possibly being optimum. This modifies the QAP, but is assured to fully enumerate all feasible solutions to the original problem. (See Section 4.3.)
- Memory requirements are reasonable. (See Section 5.4.)
- Parallelization may be naturally implemented. (See Section 6.)

In Section 2, we restate the governing equations for the QAP and describe the rules for certain operations which change elements of the cost matrix in a way that leaves all assignments to the QAP ordered with respect to their costs, thus generating a series of equivalent problems. In Section 3, we postulate the existence of a

procedure which, when carried to completion, solves the QAP from a dual perspective. We then present a Dual Procedure, which was developed experimentally, and which in practice is efficient as a lower bounding technique. Section 4 describes the branch-and-bound algorithm which was written to test out the effectiveness of the lower bounds generated by the DP. Section 5 describes the testing and evaluation of the DP as a lower bounding method. Number of nodes (partial assignments) that have to be fathomed well as total branch-and-bound run times are given. Finally, conclusions are presented in Section 6.

2. QAP Formulation

The quadratic assignment problem (QAP) is defined as follows: Given N^4 cost coefficients $C_{ijkl} \geq 0$ ($i, j, k, l = 1, 2, \dots, N$) determine an $N \times N$ matrix which is a solution

$$\mathbf{U} = [u_{ab}] \quad (1)$$

called an "assignment", so as to minimize a cost function,

$$R(\mathbf{U}) = \sum_{ijkl} C_{ijkl} u_{ij} u_{kl} \quad (2)$$

subject to the following constraints on \mathbf{U} :

$$u_{ij} = 0, 1 \quad (i, j = 1, 2, \dots, N), \quad (3)$$

$$\sum_{i=1}^N u_{ij} = 1 \quad (j = 1, 2, \dots, N), \quad (4)$$

and

$$\sum_{j=1}^N u_{ij} = 1 \quad (i = 1, 2, \dots, N) \quad (5)$$

i.e., \mathbf{U} is a permutation matrix.

Lawler [19] introduced the concept of an N^2 by N^2 solution (or assignment) matrix \mathbf{V} which is a Kronecker product of the $N \times N$ assignment matrix \mathbf{U} with itself. (Given two matrices \mathbf{A} and \mathbf{B} , a Kronecker product is formed by post-multiplying each scalar element of \mathbf{A} by \mathbf{B} , thus forming a new matrix whose dimensions are the products of the respective dimensions of \mathbf{A} and \mathbf{B} .)

That is,

$$\mathbf{V} = \mathbf{U}\mathbf{x}\mathbf{U} = \begin{matrix} u_{11}\mathbf{U} & u_{12}\mathbf{U} & & u_{1N}\mathbf{U} \\ u_{21}\mathbf{U} & u_{22}\mathbf{U} & & u_{2N}\mathbf{U} \\ \vdots & \vdots & \ddots & \vdots \\ u_{N1}\mathbf{U} & u_{N2}\mathbf{U} & & u_{NN}\mathbf{U} \end{matrix} = [v_{ijkn}] \quad (6)$$

$$\text{where } v_{ijkn} = u_{ij} u_{kn} = u_{kn} u_{ij} = v_{knij} \quad (7)$$

Lawler's Kroenecker product formulation is an apparent attempt to linearize the problem. He probably used the insight gained from this formulation to devise what is now known as the Gilmore-Lawler lower bounding procedure, which is similar to a few of the basic steps of the DP.

Equation 7 is very important to the effectiveness of the DP. It says that if an element v_{ijkn} is involved in an assignment (i.e., equal to 1) then it has a "complementary element" v_{knij} that is also equal to 1. These complementary pairs have an interesting property; they are always in two different submatrices that never occupy the same submatrix row or submatrix column. Later, we shall see that the equality between the pair creates a valuable communication between submatrices that can be exploited in improving the lower bounds of Gilmore, Lawler and others.

The \mathbf{V} matrix is a partitioned matrix whose elements are comprised of the Null matrix (all elements zero) and matrix \mathbf{U} . Furthermore, \mathbf{V} exhibits a gross pattern identical to that of matrix \mathbf{U} . An example of a \mathbf{V} matrix for $N = 3$ is shown in Figure 1.

$$\mathbf{V} = \begin{array}{ccc|ccc|ccc} \{0\} & 0 & 0 & 0 & \{0\} & 0 & 0 & 0 & \{1\} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \{ \} & 0 & 0 & 0 & \{0\} & 0 & 0 & 0 & \{0\} & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \{0\} & 0 & 0 & 0 & \{ \} & 0 & 0 & 0 & \{0\} & 0 & 0 & 0 \end{array} \quad \mathbf{U} = \begin{array}{ccc} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{array}$$

Figure 1. Example of Assignment Matrix \mathbf{V} and its \mathbf{U} matrix.

Certain elements of \mathbf{V} are always zero, specifically

$$v_{ibkb} = 0 \text{ if } i \neq k \quad (8)$$

and

$$v_{bjbn} = 0 \text{ if } j \neq n \quad (9)$$

These elements are referred to as "disallowed" elements. Elements with subscript $ijij$ ($i, j=1, 2, \dots, N$) are termed "linear-cost elements" and are shown bracketed in the figure. Observe that if there are any 1's in a submatrix $u_{ij}\mathbf{U}$ its linear-cost element is always unity and vice versa. Observe, too, that one and only one unity element may exist in each row and column of \mathbf{V} . These constraints follow directly from the definition of \mathbf{V} as the product of two permutation matrices. A linear-cost element is special in another way; it has no complementary element (because $v_{ijij} = u_{ij} u_{ij}$).

Now, suppose we arrange the N^4 cost coefficients C_{ijkn} ($i, j, k, n=1, 2, \dots, N$) in an $N^2 \times N^2$ matrix \mathbf{C} , similar to matrix \mathbf{V} and indexed in precisely the same fashion. This is demonstrated in Figure 2. In the figure the asterisks denote disallowed elements.

It can be shown that the elements of matrix \mathbf{C} which contribute to the cost $R(\mathbf{U})$ of an assignment \mathbf{U} are those and only those corresponding to the unity elements in matrix \mathbf{V} . An assignment \mathbf{V} is said to "intersect" a submatrix \mathbf{C}_{ij} of \mathbf{C} , if there are 1's in the elements of \mathbf{V} that correspond to that submatrix. An element of \mathbf{C} is said to be "involved" in an assignment \mathbf{V} , if the corresponding element of \mathbf{V} is unity. The linear-cost elements of \mathbf{C} are commonly known in the QAP literature as "linear costs".

$$\mathbf{C} = \begin{array}{c} \begin{array}{|c|c|c|} \hline \mathbf{C}_{11} & \mathbf{C}_{12} & \mathbf{C}_{13} \\ \hline \mathbf{C}_{21} & \mathbf{C}_{22} & \mathbf{C}_{23} \\ \hline \mathbf{C}_{31} & \mathbf{C}_{32} & \mathbf{C}_{33} \\ \hline \end{array} \\ \\ \begin{array}{|c|c|c|} \hline C_{1111} & & C_{1212} & & & C_{1313} \\ & C_{1122} & C_{1123} & C_{1221} & C_{1223} & C_{1321} & C_{1322} \\ & C_{1132} & C_{1133} & C_{1231} & C_{1233} & C_{1331} & C_{1332} \\ \hline * & C_{2112} & C_{2113} & C_{2211} & C_{2213} & C_{2311} & C_{2312} \\ = & C_{2121} & & C_{2222} & & & C_{2323} \\ & C_{2132} & C_{2133} & C_{2231} & C_{2233} & C_{2331} & C_{2332} \\ \hline & C_{3112} & C_{3113} & C_{3211} & C_{3213} & C_{3311} & C_{3312} \\ & C_{3122} & C_{3132} & C_{3221} & C_{3223} & C_{3321} & C_{3322} \\ C_{3131} & & & C_{3232} & & & C_{3333} \\ \hline \end{array} \end{array}$$

Figure 2. Matrix of Cost for $N = 3$.

The existence of complementary pairs opens the door to considerable flexibility in the \mathbf{C} matrix. If one element of a complementary pair is involved in an assignment, so is the other. Thus, for each complementary pair, cost can be shifted at will between the two elements. For instance, the cost of both elements can be added and placed at the first element while the cost of the second becomes zero or vice versa.

It can be shown that certain operations may be performed on $\mathbf{C} = \{C_{ijkn}\}$ which will change the cost $R(\mathbf{U})$ of assignments \mathbf{U} in such a way that all assignment costs are shifted by an identical amount, thus preserving their order with respect to cost. These operations are divided into two classes (see Grant [13] for proofs):

Class 1: Addition (or subtraction) of a constant to all feasible elements of a submatrix (\mathbf{C}_{ij}) row or column and the corresponding subtraction (or addition) of this constant from either another row or column of the submatrix or from the submatrix linear cost element.

Class 2: Addition or subtraction of a constant to all feasible elements of any row or column in matrix \mathbf{C} .

Class 1 operations maintain the cost of all assignments, but permit redistribution of element costs within a given submatrix. This follows because any submatrix involved in an assignment must itself have the form of an assignment matrix. Consequently, the transfer of cost from one submatrix row or column to another will maintain the cost of all assignments which pass through the submatrix. Furthermore, since the linear cost of an involved submatrix must also be involved in the assignment, the transfer of cost from a submatrix row or column to the linear cost, and vice versa, will also leave the cost of all assignments which intersect the submatrix unchanged.

In contrast, Class 2 operations work on the $N^2 \times N^2$ matrix level, and change the cost of all assignments by the amount added to or subtracted from the matrix row or column. This is true because one and only one cost element in a row or column of \mathbf{C} can be involved in the overall cost of an assignment.

3. A Dual to the Quadratic Assignment Problem

3.1 A Basis for a Dual formulation

If the operations on \mathbf{C} decrease the cost by an amount R' and are performed in a way that keeps the elements of \mathbf{C} non-negative, then no assignment cost can become negative and the following relationship holds:

$$R' \leq R(\mathbf{U})_{\min} \quad (10)$$

If furthermore R' can be increased until the equality holds, then the elements of the adjusted cost matrix involved in an optimum assignment would necessarily be zero. Thus a dual for the Quadratic Assignment problem can be stated as follows: Maximize the sum of downward cost shifts R' permitted by Class 1 and 2 operations, under the constraint that no cost element in \mathbf{C} is driven negative.

Hahn recognized that this is exactly what happens in the Hungarian algorithm of Munkres[15] for solving linear assignment problems. The Hungarian algorithm is a matrix reduction scheme which extracts cost from a linear cost matrix until the resulting pattern of zeros corresponds to an assignment to the LAP. When this occurs, the reduction constant yields the optimum linear assignment cost and the assignment may be obtained by inspection. The matrix reduction methods used in the Hungarian algorithm are essentially the Class 1 and Class 2 operations described above.

3.2 The Dual Procedure

The DP algorithm is based on the concept that constant amounts are subtracted from rows or columns of each (and every) submatrix and added to the linear cost element of its submatrix (Class 1 operations). Further, constant amounts are then subtracted from the rows or columns of \mathbf{C} containing only linear costs and added to the reduction constant R' (Class 2 operations). This is similar to the Gilmore-Lawler bounding method (see Gilmore [12]). Two important processes have been added, however. One is the movement of complementary costs to the submatrix where subtraction of costs is about to take place. As a result, greater transfer of submatrix costs to linear costs occurs than would have been possible, had such prior transfer (of complementary costs) not taken place. Two is that, after subtractions have been made from linear cost rows and columns, non-zero linear costs are added back to submatrix rows. This latter process, along with the movement of complementary costs between

submatrices, leaves the cost matrix \mathbf{C} dramatically rearranged. Because of this rearrangement, the process can be repeated, i.e., additional costs can be subtracted from within submatrices and moved to R' . The result is an iterative procedure that produces growth in R' with each round. This growth, while significant, diminishes with each round, so at some point it does not pay to continue the process.

At the end of each round of the DP a test is made to see if the zeros in the reduced \mathbf{C} matrix matches an assignment pattern (as is the case for the '1's in Figure 1),. If so, a terminating condition has been reached, i.e., no more costs can be subtracted from \mathbf{C} and the QAP has actually been solved. While this desirable result happens in some cases, i.e., for some small problems ($N < 7$) and for numerous larger problems ($7 \leq N \leq 16$) that arise as sub-problems of a larger QAP such as the Nugent 20, usually it does not.

The DP bears strong resemblance to well known lower-bounding techniques for branch-and-bound algorithms. The key differences in the approach are the treatment of complementary element costs and the adding of the non-zero linear costs to the submatrix rows. In general, this procedure produces tighter lower bounds than those achieved by other strategies. It also produces these bounds efficiently. This matter is covered in detail by Hahn and Grant [16].

One of us (Grant) [13] investigated the validity of the DP as a finite, convergent solution technique for the QAP. The results from this investigation indicate that, while valid for lower bound computations, the DP can not guarantee the attainment of a feasible nor an optimal assignment when operated as a stand-alone procedure. This, too, is explained in Hahn and Grant [16].

3.3 Historical Perspective

The discovery of the DP was the result of an attempt to solve the QAP from the dual perspective (see Hahn [15]). It was recognized, back in 1966, that the LAP had been solved effectively and efficiently through its dual. The hope was that the QAP could be solved using a similar approach. Lawler's linearization of the problem and his Kronecker second power representation were suitable for exploring that possibility. At first, the Hungarian method was applied to the \mathbf{C} matrix itself, but this would not result in a solution. It became apparent that individual submatrices would have to

contain solutions for the QAP to be solved, so the Hungarian method was applied to these first.

In 1966, available computer memory was limited to 32k bytes. It seemed useful then to store only one element of a complementary pair. Thus, when an individual submatrix was solved using the Hungarian method, complementary element values in $(N-1)^2$ other submatrices automatically changed. Hahn observed this and realized that there was potential for re-solving these submatrices on a second round, thus moving the dual closer to solution. Various improvements were tried, some of which worked and were kept, while others were discarded.

Though Hahn was able to solve some small problems with the method, he was unable to solve larger ones. Eventually, he gave up on the idea of solving the QAP this way, but not before he recognized that the DP produced excellent lower bounds on the original problem. He continued to use the DP to test how bounds improved, as the search space of the primal problem reduced. (See Section 4.3 to see how this works)

3.4 Improvements in the Dual Procedure

While the lower bounds achieved by the DP algorithm by itself are considerably better than those found in other general lower bounding techniques, several improvements were made in the algorithm. One notable improvement came from the judicious return of a part of the lower bound R' to the matrix of linear costs prior each round. This is done in the context of the use of simulated annealing, whereby random percentages of the lower bound R' are returned on each round; these random percentages following an exponentially decreasing annealing schedule. Some experimentation was done to optimize the selection of the exponential rate for the schedule, which affected DP algorithm speed as well as quality of lower bound achieved. The exponential rate turned out to be not very critical to performance.

4. Use of the Dual Procedure in Branch-and-bound.

The DP is utilized within a branch-and bound algorithm (called the DPB&B) as the auxiliary process for computing lower bounds. In this context the convergence limitations of the DP are not critical, and information communicated between the main algorithm and the DP serves to improve the performance of both processes.

The DPB&B algorithm is developed in a manner consistent with the DP. We follow the conventional technique of selecting a single facility-location assignment as the first (highest) level as well as subsequent levels of partial assignment. In order to implement this selection, a linear cost is chosen to be involved in the assignment. For instance, we might choose the upper-leftmost linear cost. Referring to Figure 2, this would be element C_{1111} , implying facility 1 is assigned to location 1 (i.e., $v_{1111} = 1$).

4.1 Lower bounding Strategy

Based on the selection of linear cost C_{ijij} , submatrix \mathbf{C}_{ij} is involved in the assignment. The remainder of the submatrices in the row and column containing submatrix \mathbf{C}_{ij} disappear (as they cannot be involved in the assignment) and the problem is thus reduced to a QAP of size $N-1$. One consequence of this reduction is that any symmetries in the original problem disappear as well. It turns out that one row and one column likewise disappear from each submatrix of the original problem, with the exception of the original submatrix \mathbf{C}_{ij} , which remains $N-1$ in size. To complete the formulation of the newly formed $N-1$ problem, this submatrix is added (by simple matrix addition) to the now size $N-1$ matrix of linear costs.

It is the application of the DP lower bounding calculation on the $N-1$ size problem that attempts to fathom a partial assignment postulated by the selection of linear cost C_{ijij} . By fathoming, one calculates a lower bound and tests it against the best known upper bound. If the best known upper bound is exceeded, the partial assignment is eliminated from the problem.

You may recall in Section 3, the DP moves costs out of the \mathbf{C} matrix into a lower bound value, leaving a modified matrix \mathbf{C}' . For subsequent branch-and-bound operations along a given partial assignment path, our strategy is to take advantage of this fact and to use this reduced cost matrix \mathbf{C}' for setting up subsequent sub-problems deeper into the tree. Thus, lower bounds are calculated not from the original problem, but from the sub-problems that were already processed by the DP at earlier (higher) levels of partial assignment. Using the modified matrix \mathbf{C}' of each of these sub-problems has the additional benefit that the sub-problem is brought closer to dual solution, making it more likely that a sub-problem will actually be solved by the DP.

In implementing the DP as a lower bounding calculation, the choice of m (number of dual iterations) at a given partial assignment became a dynamic decision, based on the closeness of the achieved lower bound after a fixed number of iterations. It had been determined experimentally that lower bounding performance early in the procedure was an excellent (though not infallible) predictor of the lower bound that could eventually be reached. Thus, if after a small number of iterations at the current partial assignment, a threshold upper bound were not reached, the lower bounding attempt would stop and the algorithm would proceed to make an additional assignment. The choice of this threshold was determined experimentally.

4.2 Partial Assignment (i.e., Branching) Strategy

When fathoming at the level of a single assignments fails, we make an additional independent assignment (i.e., choose linear cost C_{knkn}), thus reducing the original QAP to size $N-2$. There are now two submatrices (C_{ij} and C_{kn}) from the original problem that are involved in the newly formed problem of size $N-2$. This time, the remaining original submatrices loses two rows and two columns each, as does the original linear cost matrix. Both C_{ij} and C_{kn} are now reduced in size by 1 and added (by simple matrix addition) to the new linear cost matrix. An interesting consequence of this second level of partial assignment is that the element C_{ikjn} is necessarily involved in the assignment.

In solving the Nugent 15 [22] and the Elshafei 19 [10] problems, many of the partial assignments at the second level are eliminated by the DP. In the Nugent 20, fewer partial assignments are eliminated at this level, though a large percentage of those partial assignments within the submatrices containing the optimum assignment are nonetheless eliminated easily.

If fathoming fails to eliminate a partial assignment at the second level, the algorithm continues to reduce the problem size by one level at a time. In our experience with test instances of size 20 and less, it was never necessary to go beyond the 10th level, as fathoming always succeeded at this level. In fact, when solving the Nugent size 20, it was only once necessary to go to the tenth level. At the first implementation of the DPB&B the Nugent size 20 was solved without having to go

beyond the eighth level. However, subsequent speedups in the lower bounding calculation resulted in the single instance where a greater depth was required.

4.3 Search Strategy

The search strategy for selecting the next node is a simple depth first strategy. If fathoming a given node is unsuccessful, an additional partial assignment is made, thus increasing the depth into the tree by one. If a node has been fathomed successfully, the next available node is selected. Partial assignments are selected from top-left to bottom right, i.e., linear cost selection is C_{1111} , C_{1212} , C_{1313} , etc.

When the algorithm accumulates sufficient fathoming information such that permanent (or temporary) decisions can be made on the assignment matrix, these decisions are communicated to the DP and serve to improve the lower bound calculations. These decisions involve deciding that certain elements of the assignment matrix v_{ijkn} are zeros, and are recorded in the modified cost matrix \mathbf{C}' by setting the corresponding element costs to a very large value, denoted 'GREAT'. This effectively bars the 'decided 0' elements from inclusion in a DP solution, and focuses the reduction efforts on those elements still eligible for consideration. If instead a 'decide 1' element is determined, that decision is recorded in the DP cost matrix by adjusting the costs of all elements subsequently restricted from an assignment (i.e., v_{ijkn} implied to be 0 as a consequence of this 'decide 1') to the value 'GREAT'. In either case, the communication of permanent decisions to the DP generally permits additional cost to be extracted from the matrix, resulting in an improved lower bound.

5. Dual Procedure Branch-and-Bound Testing and Evaluation

5.1 Testing methodology

The results of testing of the DPB&B are given in this Section along with a comparison of results recently reported by other researchers. The experiments were run on a SPARC 10 standalone workstation with a single 75 MHz SuperSparc CPU. The program was written entirely in FORTRAN 77.

As a separate first step in the implementation of the algorithm, the DP is applied to the original QAP and the resulting reduced cost matrix \mathbf{C}' becomes the starting point for the branch-and-bound algorithm. Recall from the discussion in Section 3 that the

reduced cost matrix \mathbf{C}' is entirely equivalent to the original problem, since all assignments have retained their order and have been equally shifted by the fixed amount shown in the lower bound R' . Thus, if R' is added to the cost of any assignment cost in the restructured problem, its cost in the original problem is determined.

While the DPB&B algorithm itself was relatively naive, we did take advantage of the symmetries inherent in Nugent's test instances. The algorithm used those symmetries to eliminate "mirror image" partial assignments and thus reduced search efforts. For these tests, the starting upper bound was taken to be the cost of the best known assignment to the QAP involved. While this might be considered cheating, it can be shown that starting with a poorer upper bound would not significantly change the relative run times.

5.2 Effectiveness of DP Lower bounds in a Branch-and-bound Algorithm

An important measure of the effectiveness of a lower bounding procedure is the tightness of the lower bounds that it produces. When the lower bounds are tight, partial assignments are eliminated when only a few have been made. In the terminology of branch-and-bound techniques, each partial assignment is termed a node of the problem. The fewer the number of nodes that must be fathomed by the algorithm, the better its performance.

For comparison of different codes, several standard instances have been selected from QAPLIB [7]. These include some from the famous set of Nugent instances [22] known for their difficulty, where problems of size 6, 8, 12, 15, 20 and 30 can be found. Several others from QAPLIB are also included. Table 1 lists, for these instances, the number of nodes fathomed by the DPB&B and by two other competing algorithm codes: the parallel branch-and-bound algorithm of Clausen and Perregaard [9] and the variance-reduction branch-and-bound technique of Pardalos, et. al [24].

In Table 1 and Table 2, the data for the Clausen and Perregaard experiments for the Hadley, Roucairol and Nugent 21 and 22 instances were from an e-mail message sent by Clausen in June of 1996. In all but the Elshafei and Nugent 8 instances, the DPB&B needs to fathom far fewer nodes.

Table 1 Number of Nodes Fathomed by Competing Algorithms			
QAP Instance	DPB&B	Clausen	Pardalos
Nugent 8	131	32	895
Nugent 12	380	3,359	49,063
Nugent 15	2,203	105,773	1,794,507
Hadley 16	13,808	18,770,885	N/A
Hadley 18	197,487	761,452,218	N/A
Elshafei 19	743	471	N/A
Nugent 20	724,289	360,148,026	N/A
Nugent 21	3,192,565	3,631,929,368	N/A
Nugent 22	10,768,366	48,538,844,413	N/A
Roucairol 20	2,090,862	2,161,665,137	N/A

N/A = not available

5.3 Comparison of DPB&B Run Times

Table 2 gives the run times of the DPB&B to the solution of the same ten instances and again compares these to the recent results by Clausen and Perregaard [9] and Pardalos, et al [23].

Table 2 Branch-and-bound run time (in equiv. CPU mins)			
QAP Instance	DPB&B	Clausen	Pardalos
Nugent 8	0.03	0.016	0.005
Nugent 12	0.3	0.07	0.57
Nugent 15	3.8	2.3	33.3
Hadley 16	22.3	7,140	N/A
Hadley 18	561.1	6,188	N/A
Elshafei 19	3.8	0.12	N/A
Nugent 20	2,665	13,524	N/A
Nugent 21	10,375	44,982	N/A
Nugent 22	30,207	805,140	N/A
Nugent 24	**360,000	N/A	N/A
Roucairol 20	16,501	26,908	N/A

N/A = not available **estimated run time (currently running)

Clausen and Perregaard's experiments were run on parallel machines. Therefore, we estimated the relation between parallel run time and the time that their algorithm would have taken on a single processor machine and entered the latter time in Table 2. For a 16 processor machine we used a run time factor of 14, for a 32 processor machine we used 28, for 48 processors we used 42. For the Nugent 22 the number of processors varied between (48...96) so we took a guess at 72 processors and gave that a run time factor of 63. Checking our estimated factors with Clausen, they appear to be conservative, for this comparison.

The DPB&B is generally faster than the other three methods and is 26.6 times faster than the only other known result for the very difficult Nugent 22 instance. The DPB&B shows further promise, in that its rate of growth in runtime with Nugent problem size is less than that for either of the other methods, as can be seen from Figure 3.

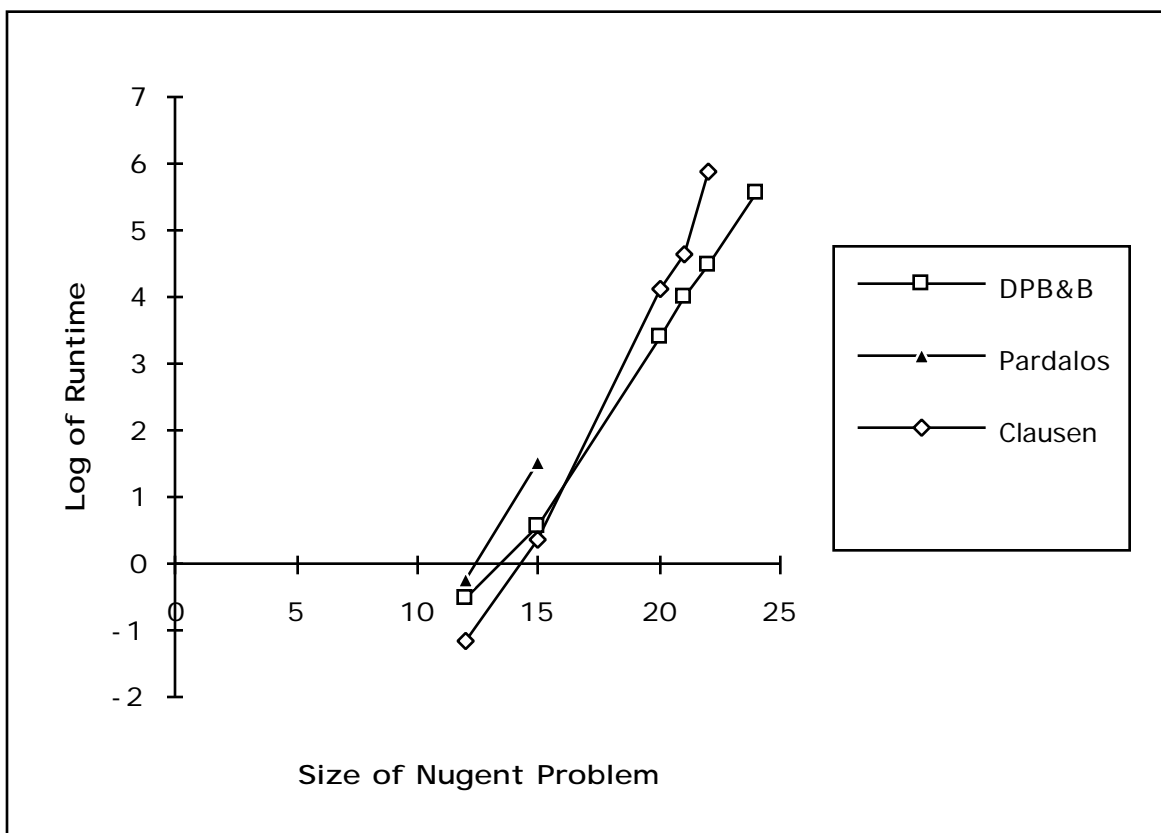


Figure 3 Comparative Run Times for Branch-and-Bound Algorithms

It was observed, in the above experiments, that the DP produces complete assignments (i.e., terminates with a pattern of zeros that constitute an assignment),

without having to progress through all the partial assignments from an inner node of the tree to its outermost nodes. This happens at various levels of partial assignment, but usually between a node where four partial assignments have been made and a node where eight partial assignments have been made. It is hardly ever necessary to go beyond the level where ten partial assignments have been made in order to achieve an assignment this way.

When the DPB&B procedure attempts to fathom partial assignments in the submatrix that is involved in an optimum assignment, it always terminates with a pattern of zeros that constitute that optimum assignment. (As the upper bound in the experiment is the optimum value of the objective function, the assignment must be optimum). The optimum assignment is generated, in fact, many times during the process of fathoming within that submatrix, once for each element that would be involved in the optimum assignment.

5.4 Memory Requirements

While speed of a QAP algorithm is the overriding concern of most practitioners, it is a well known fact that speed and memory requirements can often be traded. Some algorithms have indeed had problems with excessive memory used by arrays, tables and maps as well as that needed for tracking eliminated partial assignments. Just such a problem was experienced in Pardalos, et al. [23] while solving the Nugent 15. The growth in memory required to keep track of eliminated partial assignments exceeded the memory that was allocated.

As mentioned in Section 4.3, the fathoming decisions are recorded directly in the cost matrix \mathbf{C} (or rather the reduced cost matrix \mathbf{C}'). Therefore, there is no need to allocate memory for keeping track of eliminated partial assignments.

Table 3 lists the random access memory (RAM) requirements of the DPB&B as a function of problem size. The amounts listed in the Table comprise all memory requirements that depend upon problem size, including all variables, arrays, tables and maps. It does not include memory for the algorithm code, which of course does not change with problem size.

Our experiments were run on a machine having only 48 Mbytes of RAM so that problems of size 25 or larger required some swapping of program memory with the

hard disk. This turns out to have very little impact on the run times, as actual swapping is minimal.

Problem Size	RAM Required (Bytes)
8	285,472
12	1,894,412
15	5,464,616
16	7,438,452
18	13,084,680
20	21,707,372
21	27,445,080
22	34,319,616
24	52,100,292
25	63,343,392
30	150,776,112
32	204,398,516

5.5 Current Efforts

We are currently running the DPB&B on a Nugent size 24 instance which was derived from the very difficult to solve Nugent 30. In a run of four weeks on a SPARC 10, 75 MHz single processor workstation, it was possible to ascertain that it would take somewhere between 5 and 6 months to confirm the best assignment. At the same time, we are exploring with Clausen and Karisch at the University of Copenhagen the possibility of parallelizing our algorithm. We hope to be in a position to report new results in the near future.

6. Conclusions

We have developed a procedure for solving the QAP which is based upon a mathematical dual of the problem and which has roots in the Hungarian method for solving the linear assignment problem. A unique and very important aspect of the DP is that at each stage, the QAP is restructured as fully equivalent to the original QAP in a manner that brings it closer to solution. In some cases the QAP is actually solved

exactly. Just as importantly, at each stage of the DP, a high quality lower bound is economically calculated which is ideally suited to a branch-and-bound solution of the QAP. We have embedded the DP in a branch-and-bound algorithm which is also unique in many respects and have solved the Nugent size 22 QAP. Only one other team of researchers (Clausen and Perregaarde [9]) have been able to solve that problem exactly.

The performance of a DPB&B would obviously benefit by the use of a multiprocessor-based computer. A parallel approach for a branch-and-bound algorithm was experimented with by Roucairol [26] with promising results. Clausen and Perregaard were the first to solve the hitherto unsolved Nugent 20, 21 and 22 instances with such a technique. For the DP, the submatrix LAPs could be solved in parallel, thus achieving a speed improvement proportional to the number of processors used. One problem might arise as a result of the tight coupling of complementary cost pairs; the simultaneous Hungarian solution of two submatrices containing a common cost pair could be cumbersome. Fortunately, any submatrices residing in a the same submatrix row or in a the same submatrix column can be solved simultaneously, as they have no elements in common. Thus, parallelization of order N is always assured.

In the last five years, considerable progress has been made by the research community on heuristic methods and on mathematical lower bounding techniques which give branch-and-bound techniques the potential for solving larger and larger problems. Additionally, several authors have emphasized the need for complementary techniques to solve the more difficult problems which a single technique cannot even attempt to solve, such as VLSI design (see Areibi and Vannelli) [1]. The DP is just such a complementary technique. It can be incorporated in a variety of heuristic and exact methods. Furthermore, the individual processes described herein may enhance other QAP and related problem algorithms.

References

- [1] Areibi, S and Vannelli, A, "Advanced Search Techniques for Circuit Partitioning, In *Quadratic Assignment and Related Problems*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, edited by P. Pardalos and H. Wolkowicz, vol. 16, (1994): 77-99.
- [2] Assad, A. A., and Xu, W, "On Lower Bounds for a Class of Quadratic 0,1 Programs," *Operations Research Letters*, vol. 4 (1985): 175-180.
- [3] Bazaraa, M. S., and Kirca, O. : "A Branch-and-Bound-Based Heuristic for Solving the Quadratic Assignment Problem", *Naval Research Logistics Quarterly*, vol. 30, no. 1 (1983): 287-304
- [4] Burkard, R. E. "A Heuristic for Quadratic Boolean Programs with Applications to Quadratic Assignment Problems", *European Journal of Operational Research*, Vol. 13, (1983): 374-386.
- [5] Burkard, R. E. : "Quadratic Assignment Problems", *European Journal of Operational Research*, vol. 15, no. 3 (Mar. 1984): 283-289.
- [6] Burkard, R. E., "Locations with Spatial Interactions: The Quadratic Assignment Problem," in *Discrete Location Theory*, P. B. Mirchandani and R. L. Francis, eds., John Wiley & Sons (1991): 387-437.
- [7] Burkard, R. E., Karisch, S. E. and Rendl, F., "QAPLIB - A Quadratic Assignment Problem Library," *European Journal of Operational Research*, vol. 55, no. 99 (1991): 115-119.
- [8] Carraresi, P, and Malucelli, F, "A New Lower Bound for the Quadratic Assignment Problem," *Operations Research*, vol. 40 (1992) Supple. no. 1: S22-S27.
- [9] Clausen, J. , and Perregaard, M. "Solving Large Quadratic Assignment Problems in Parallel", June 30, 1995, : accepted for publication in *Computational Optimization and Applications*.

- [10] Elshafei, A. : "Hospital Layout as a Quadratic Assignment Problem", *Operations Research Quarterly*, vol. 28, no. 1 (1977): 167-179.
- [11] Frieze, A.M., and Yadegar, J. : "On the Quadratic Assignment Problem", *Discrete Applied Mathematics*", vol. 5, no. 1 (Jan. 1983): 89-98.
- [12] Gilmore, P.C. : "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem", *Journal of the Society of Industrial and Applied Mathematics*, vol. 10, no. 2 (1962): 305-313.
- [13] Grant, T. L., "An Evaluation and Analysis of the Resolvent Sequence Method for Solving the Quadratic Assignment Problem", Master's Thesis, University of Pennsylvania, 1989.
- [14] Hadley, S., Rendl, F., and Wolkowicz, H., "Bounds for the Quadratic Assignment Problem Using Continuous Optimization Techniques", in *Integer Programming and Combinatorial Optimization*, University of Waterloo Press (1990): 237-248.
- [15] Hahn, P.M., "Minimization of Cost in Assignment of Codes to Data Transmission", Ph.D. Dissertation, University of Pennsylvania, 1968.
- [16] Hahn, P.M., and Grant, T. L., "Lower Bounds for the Quadratic Assignment Problem Based Upon a Dual Formulation", Being considered for publication by the Operations Research Society of America.
- [17] House, R. W., Nelson, L. D., Rado, T. : "Computer Studies of a Certain Class of Linear Integer Programs", *Recent Advances in Optimization Techniques*, A. Lavi and T. P. Vogl, eds., John Wiley and Sons (1965).
- [18] Koopmans, T. C., and Beckmann, M. J., "Assignment Problems and the Location of Economic Activities," *Econometrica*, vol. 25 (1957): 53-76.
- [19] Lawler, E. L. : "The Quadratic Assignment Problem", *Management Science*, vol. 9 (1963): 586-599.

- [20] Li, Y., Pardalos, P., Resende, M., "A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 16 (1994).
- [21] Munkres, M., "Algorithms for the Assignment and Transportation Problems", *Journal of the Society of Industrial and Applied Mathematics*, vol. 5, no. 1 (March 1957): 32-38.
- [22] Nugent, C. E., Vollman, T. E., Ruml, J. : "An Experimental Comparison of Techniques for the Assignment of Facilities to Locations", *Operations Research*, vol. 16 (Jan.-Feb. 1968): 150-173.
- [23] Pardalos, P.M., Crouse, J. V., "A Parallel Algorithm for the Quadratic Assignment Problem", *Proceedings of the 1989 Supercomputing Conference* , ACM Press, (1989): 351-360.
- [24] Pardalos, P. M., Ramakrishnan, K. G, Resende, M. G. C, and Li, Y., "Implementation of a Variance Reduction Based Lower Bound in a Branch-and-bound Algorithm for the Quadratic Assignment Problem", unpublished.
- [25] Roucairol, C : "A Reduction Method for the Quadratic Assignment Problem", *Operations Research Verfahren-Methods of Operations Research*, vol. 32 (1979): 185-187.
- [26] Roucairol, C. : "A Parallel Branch-and-bound Algorithm for the Quadratic Assignment Problem", *Discrete Applied Mathematics*, vol. 18, no. 2 (Nov. 1987): 211-225.
- [27] Sahni, S. and Teogilo, G., "P-Complete Approximation Problems," *Journal Assoc. Computing Machinery*, vol. 23, no. 3 (1976): 555-565.