[31] M. Resende, K. Ramakrishnan, and Z. Drezner, *Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming*, tech. rep., AT&T Bell Laboratories, Murray Hill, NJ 07974, 1994.

[32] C. Roucairol, *A parallel branch and bound algorithm for the quadratic assignment problem*, Discrete Applied Mathematics, 18 (1987), pp. 211–225.

[33] J. Skorin-Kapov, *Tabu search applied to the quadratic assignment problem*, ORSA Journal on Computing, 2 (1990), pp. 33–45.

[34] E. Taillard, *Robust tabu search for the quadratic assignment problem*, Parallel Computing, 17 (1991), pp. 443–455.

[35] Thonemann, U.W. and Bölte, A.M., *An improved simulated annealing algorithm for the quadratic assignment problem*, tech. rep., School of Business, Dept. of Production and Operations Research, University of Paderborn, Germany, 1994.

[20] Y. Li, P. Pardalos, and M. Resende, *A greedy randomized adaptive search procedure for the quadratic assignment problem*, in Quadratic assignment and related problems, P. Pardalos and H. Wolkowicz, eds., vol. 16 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1994, pp. 237–261.

[21] T. Mautor and C. Roucairol, *Difficulties of exact methods for solving the quadratic assignment problem*, in Quadratic assignment and related problems, P. Pardalos and H. Wolkowicz, eds., vol. 16 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1994, pp. 263–274.

[22] ——, *A new exact algorithm for the solution of quadratic assignment problems*, Discrete Applied Mathematics, (1994). To appear.

[23] H. Mawengkang and B. Murtagh, *Solving nonlinear integer programs with large-scale optimization software*, Annals of Operations Research, 5 (1985/6), pp. 425–437.

[24] E. McCormik, *Human factors engineering*, McGraw-Hill, New York, 1970.

[25] J. E. Mitchell, *Interior point algorithms for integer programming*, tech. rep., Rensselaer Polytechnic Institute, June 1994. To appear in *Advances in linear and integer programming*, Oxford University Press, 1995.

[26] B. Murtagh, T. Jefferson, and V. Sornprasit, *A heuristic procedure for solving the quadratic assignment problem*, European Journal of Operational Research, 9 (1982), pp. 71–76.

[27] P. Pardalos and J. Crouse, *A parallel algorithm for the quadratic assignment problem*, in Proceedings of the Supercomputing 1989 Conference, ACM Press, 1989, pp. 351–360.

[28] P. Pardalos, K. Ramakrishnan, M. Resende, and Y. Li, *Implementation of a variance reduction based lower bound in a branch and bound algorithm for the quadratic assignment problem*, tech. rep., AT&T Bell Laboratories, Murray Hill, NJ 07974, 1994.

[29] P. Pardalos and H. Wolkowicz, eds., *Quadratic assignment and related problems*, vol. 16 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1994.

[30] M. Resende, P. Pardalos, and Y. Li, *FORTRAN subroutines for approximate solution of dense quadratic assignment problems using GRASP*, tech. rep., AT&T Bell Laboratories, Murray Hill, NJ, 1994. To appear in *ACM Transactions on Mathematical Software.*

State University, Fullerton, CA 92634, 1994. To appear in *Computational Optimization & Applications.*

[8] C. Edwards, *A branch and bound algorithm for the Koopmans-Beckman quadratic assignment problem*, Mathematical Programming Study, 13 (1980), pp. 35–52.

[9] C. Fleurent and J. Ferland, *Genetic hybrids for the quadratic assignment problem*, in Quadratic assignment and related problems, P. Pardalos and H. Wolkowicz, eds., vol. 16 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1994, pp. 173–187.

[10] R. Fourer, D. Gay, and B. Kernighan, *AMPL – A modeling language for mathematical programming*, The Scientific Press, South San Francisco, CA, 1993.

[11] R. Francis and J. White, *Facility layout and location*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

[12] P. Gilmore, *Optimal and suboptimal algorithms for the quadratic assignment problem*, J. SIAM, 10 (1962), pp. 305–313.

[13] L. Hubert, *Assignment methods in combinatorial data analysis*, Marcel Dekker, Inc., New York, NY 10016, 1987.

[14] T. Ibaraki, *Theoretical comparisons of search strategies in branch-and-bound algorithms*, International Journal of Computer and Information Sciences, 5 (1976), pp. 315–344.

[15] N. Karmarkar and K. Ramakrishnan, *Computational results of an interior point algorithm for large scale linear programming*, Mathematical Programming, 52 (1991), pp. 555–586.

[16] T. Koopmans and M. Beckmann, *Assignment problems and the location of economic activities*, Econometrica, 25 (1957), pp. 53–76.

[17] J. Krarup and P. Pruzan, *Computer-aided layout design*, Mathematical Programming Study, 9 (1978), pp. 75–94.

[18] E. Lawler, *The quadratic assignment problem*, Management Science, 9 (1963), pp. 586–599.

[19] Y. Li, P. Pardalos, K. Ramakrishnan, and M. Resende, *Lower bounds for the quadratic assignment problem*, Annals of Operations Research, 50 (1994), pp. 387–410.

nodes. As the size of the QAP grew the CPU time ratio of the time taken by the new code to the time taken by the GLB based code decreased dramatically. However, for problems of the dimensions considered, it is still faster to use the GLB-based branch and bound algorithm.

In a forthcoming paper, we expect to produce examples for which the new approach, besides scanning much fewer nodes, is also much faster. For this, we plan several extensions to our implementation. Since linear programs on different branches of the search tree are essentially independent of each other, it is possible to solve them in parallel. We are implementing a distributed algorithm that solves different linear programs on different processors, broadcasting the value of a new upper bound whenever one is found. In the version of the ADP code used in this paper, the algorithm does not have the capability to do warm starts, i.e. start from an advanced solution. The linear programs are always solved from the start, even when one LP varies from the other by a single column in the dual program. We are implementing a version of ADP that can start from a warm solution. With this, we expect to speed up the solution process significantly.

References

[1] S. Bokhari, *Assignment problems in parallel and distributed computing*, The Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, Boston/Dordrecht/Lancaster, 1987.

[2] B. Borchers and J. E. Mitchell, *Using an interior point method in a branch and bound algorithm for integer programming*, tech. rep., Rensselaer Polytechnic Institute, July 1992.

[3] R. Burkard and U. Derigs, *Assignment and matching problems: Solution methods with Fortran programs*, vol. 184 of Lecture Notes in Economics and Mathematical Systems, Springer, Berlin, 1980.

[4] R. Burkard, S. Karisch, and F. Rendl, *QAPLIB – A quadratic assignment problem library*, European Journal of Operational Research, 55 (1991), pp. 115–119. Updated version – Feb. 1994.

[5] R. Burkard and F. Rendl, *A thermodynamically motivated simulation procedure for combinatorial optimization problems*, European Journal of Operational Research, 17 (1984), pp. 169–174.

[6] J. Clausen. Announcement on Discrete Mathematics and Algorithms Network (DIMANET), Feb. 1994.

[7] Z. Drezner, *Lower bounds based on linear programming for the quadratic assignment problem*, tech. rep., Dept. of Management Science, California

Table 5: CPU time ratios for QAPs of consecutive sizes

n	t_n	t_n/t_{n-1}
2	0.93	-
3	1.08	1.09
4	1.35	1.09
5	1.95	1.51
6	3.08	1.47
7	5.03	1.54
8	7.39	1.66
9	12.34	1.54
10	21.04	1.52
11	39.13	1.49
12	49.86	1.67
13	126.51	1.54
14	280.44	1.53

for $n = 2, 3, \ldots, 14$, where t_n is the average time taken to solve a problem of dimension n. As can be observed, for problems in this experiment, it takes slighltly over 50% longer to solve a problem of dimension n as it takes to solve a problem of dimension $n - 1$.

6 Concluding Remarks

In this paper, we presented implementation details and computational results of a new branch and bound algorithm for solving the quadratic assignment problem. The branch and bound algorithm uses a branching scheme discussed in [28] with a lower bounding scheme that uses linear programming described in [31]. An efficient implementation [15] of an interior point algorithm is used to compute the lower bounds. The use of interior point methods in the context of branch and bound methods for integer programming was first described by Mitchell and Borchers [2]. See Mitchell [25] for a survey on interior point algorithms for integer programming.

Our implementation successfully solved to optimality all instances of QAPLIB [4] with dimension $n \leq 15$. The main observation is that the lower bounds are good, resulting in the search of very few branch and bound search tree nodes compared to the same branching scheme using the classical Gilmore-Lawler lower bound [12, 18, 28]. The number of nodes scanned grew less than a cubic function of n, the dimension of the QAP. The instance with the largest number of nodes (`nug15` of dimension $n = 15$) required the solution of 1195 linear programs. Other instances of the same size required much fewer (as few as 15)

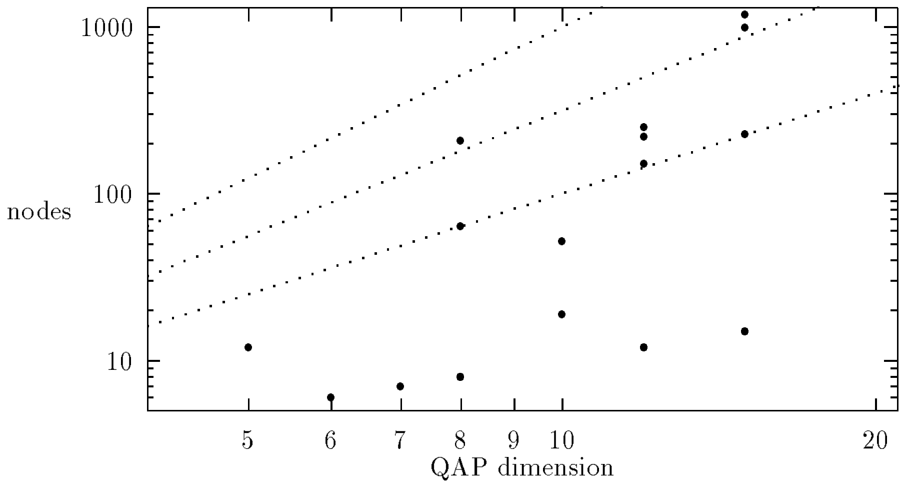

Figure 3: B&B tree nodes scanned and functions n^2, $n^{2.5}$, and n^3

- The code solved all 24 instances of QAPLIB of dimension less than or equal to 15.

- Compared with the branch and bound algorithm using the classical Gilmore-Lawler lower bound, the number of branch and bound search tree nodes examined by the algorithm is small and grows slowly as a function of the dimension n of the QAP.

- Because no level 0 nodes (the QAP linear programming relaxation for the original problem) were done, the minimum number of nodes examined by the algorithm is n, the number of level 1 nodes in the tree. Level 0 nodes were solved for all the instances solved in this paper in [31]. In 10 of the 24 problems solved in this paper (`nug05`, `nug06`, `nug07`, `esc08c`, `esc08f`, `chr12a`, `chr12b`, `chr12c`, `chr15b`, and `chr15c`) the level 0 lower bound produced was tight and thus all but one of the problems could be solved at level 0, since the inital upper bounds produced were optimal. The only exception is `nug05` for which the GRASP upper bound initially produced is not optimal.

- The times to solve the linear programming relaxations grows as indicated in Figure 2. Table 5 shows the ratios t_n/t_{n-1} of average running times

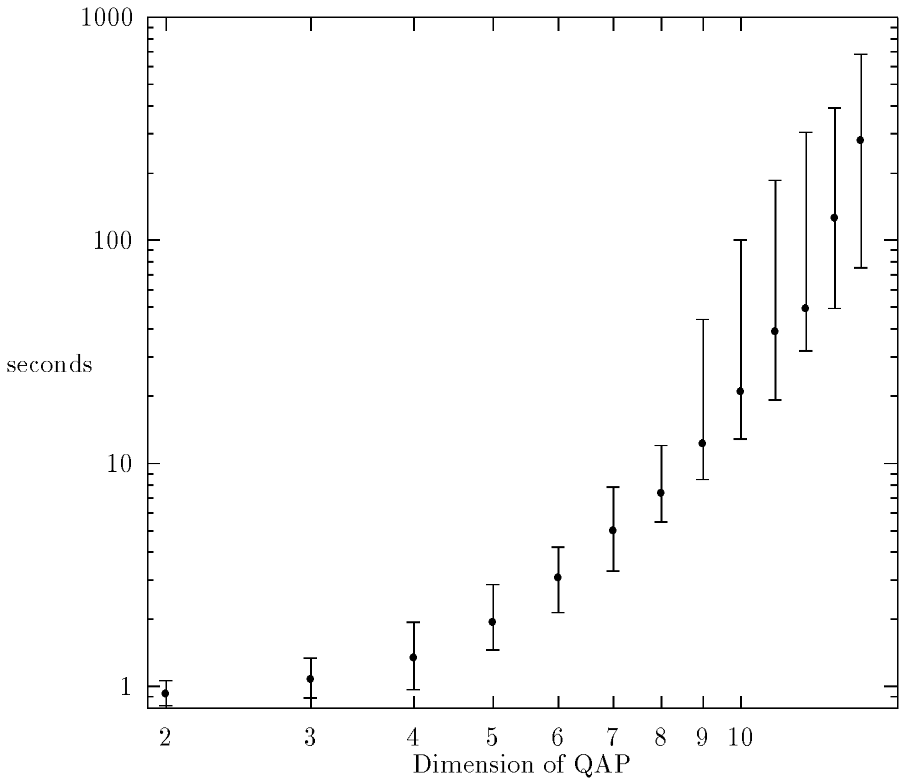

Figure 2: ADP LP solver CPU times

Table 4: QAP test instances: B&B tree search

	nodes of B&B tree			percentage of nodes of level				
problem	scan	good	bad	1	2	3	4	≥ 5
nug05	14	10	4	35.7	28.6	21.4	14.3	0.0
nug06	6	6	0	100.0	0.0	0.0	0.0	0.0
nug07	7	7	0	100.0	0.0	0.0	0.0	0.0
nug08	8	8	0	100.0	0.0	0.0	0.0	0.0
nug12	220	200	20	5.5	45.0	45.5	4.1	0.0
nug15	1195	1103	92	1.3	17.6	56.6	21.1	3.5
scr10	19	18	1	52.6	47.4	0.0	0.0	0.0
scr12	252	228	24	4.8	43.7	23.8	21.4	6.3
scr15	228	211	17	6.6	49.1	11.4	10.5	22.4
rou10	54	46	8	18.5	16.7	14.8	13.0	37.0
rou12	154	137	17	7.8	57.1	6.5	5.8	22.7
rou15	991	912	79	1.5	21.2	69.5	1.2	6.6
esc08a	8	8	0	100.0	0.0	0.0	0.0	0.0
esc08b	208	176	32	3.8	26.9	69.2	0.0	0.0
esc08c	8	8	0	100.0	0.0	0.0	0.0	0.0
esc08d	8	8	0	100.0	0.0	0.0	0.0	0.0
esc08e	64	56	8	12.5	87.5	0.0	0.0	0.0
esc08f	8	8	0	100.0	0.0	0.0	0.0	0.0
chr12a	12	12	0	100.0	0.0	0.0	0.0	0.0
chr12b	12	12	0	100.0	0.0	0.0	0.0	0.0
chr12c	12	12	0	100.0	0.0	0.0	0.0	0.0
chr15a	15	15	0	100.0	0.0	0.0	0.0	0.0
chr15b	15	15	0	100.0	0.0	0.0	0.0	0.0
chr15c	15	15	0	100.0	0.0	0.0	0.0	0.0

Table 3: QAP test instances: LP-based vs. GLB-based B&B algorithms

		LP-based B&B		GLB-based B&B		time	nodes
problem	dim	nodes	time	nodes	time	ratio	ratio
nug05	5	12	11.7	44	0.1	117.0	3.7
nug06	6	6	9.5	82	0.1	95.0	13.7
nug07	7	7	16.6	115	0.1	166.0	16.4
nug08	8	8	35.1	895	0.2	175.5	111.9
nug12	12	220	5238.2	49063	14.6	358.8	223.0
nug15	15	1195	87085.7	1794507	912.4	95.4	1501.7
scr10	10	19	202.1	1494	0.6	336.8	78.6
scr12	12	252	5118.7	12918	4.8	1066.4	51.3
scr15	15	228	3043.3	506360	274.7	11.1	2220.9
rou10	10	52	275.7	2683	0.8	344.6	51.6
rou12	12	152	2715.9	37982	12.3	220.8	249.9
rou15	15	991	30811.7	4846805	2240.3	13.8	4890.8
esc08a	8	8	37.4	57464	7.0	5.3	7183.0
esc08b	8	208	491.1	7352	0.7	701.6	35.3
esc08c	8	8	42.7	2552	0.3	142.3	319.0
esc08d	8	8	38.1	2216	0.3	127.0	277.0
esc08e	8	64	251.0	10376	1.0	251.0	162.1
esc08f	8	8	37.6	1520	0.3	125.3	190.0
chr12a	12	12	312.0	672	0.7	445.7	56.0
chr12b	12	12	289.4	318	0.6	482.3	26.5
chr12c	12	12	386.1	3214	1.5	257.4	267.8
chr15a	15	15	1495.9	413825	235.5	6.4	27588.3
chr15b	15	15	1831.9	396255	217.8	8.4	26417.0
chr15c	15	15	1908.5	428722	240.0	8.0	28581.5

explored. If the data is integer, as is the case for all of the QAPLIB instances considered in this paper, the dual interior solution can be rounded up to check for termination. This is done in our code. A detailed description of the ADP code as used in this application is given in Resende, Ramakrishnan, and Drezner [31].

5 Computational Results

In this section, we present preliminary experimental results with our code. The new branch and bound algorithm is tested on a set of standard QAP instances from the QAPLIB. We compare the new algorithm with an algorithm that is identical, except that it uses the classical Gilmore-Lawler lower bound in place of the LP-based bounds.

The experiments were conducted on a Silicon Graphics (SGI) Challenge (150 MHz MIPS R4400 processor, 1920 Mbytes of main memory, 16 Kbytes of data cache, and 16 Kbytes of instruction cache). The branch and bound control module and the GRASP are implemented in Fortran and compiled with the `f77` compiler using compiler flags `-O2 -Olimit 800`. The interior point solver ADP is written in C and was compiled with the `cc` compiler using compiler flags `-O2 -Olimit 800`.

We tested the codes on several instances from the QAP library QAPLIB. Table 3 summarizes the runs on both algorithms. For each instance it displays the name and dimension of the problem, as well as the solution times and number of branch and bound search tree nodes examined by each of the algorithms. The ratio of CPU times is also displayed.

The number of GRASP iterations was set to 100,000 for all runs.

Table 4 shows statistics for the LP-based algorithm. For each run, the table lists the number of nodes examined, the number of nodes on which the lower bound obtained was greater than the best upper bound at that moment, the number of nodes on which the lower bound obtained was less than or equal to the best upper bound at that moment, and the percentage of nodes examined that were of levels 1, 2, 3, 4, and 5 or greater.

Figure 2 shows the CPU times taken by the ADP solver to solve QAP lower bound linear programming relaxations as a function of QAP dimension. The figure shows the average time with a point and indicates the interval of running time values. Note that QAP instances of different sizes are solved during the branch and bound solution process. As the search goes deeper into the tree, the algorithm must compute lower bounds for smaller QAPs.

Figure 3 shows number of of the branch and bound search tree scanned by the algorithm as a function of the dimension (n) of the QAP being solved. The figure also displays the functions n^2, $n^{2.5}$, and n^3 to give an idea as to how fast the growth rate is.

We make the following remarks regarding the computational results.

As an example of the branch and bound algorithm, consider the QAPLIB instance `nug05`. The iterations of the branch and bound algorithm are summarized in Table 2. The GRASP approximation algorithm produced a solution (UB) having cost 52. The branch and bound algorithm examined 14 nodes of the search tree. In the first five nodes, each facility was fixed to location 1 and the lower bounds of each branch computed. The lower bounds corresponding to branches rooted at nodes 1 through 4 were all greater than or equal to the upper bound, and thus those branches of the tree could be pruned. At node 6 a level-2 branching begins with a lower bound less than the upper bound produced at node 7. Deeper branchings are done at nodes 8, 11, and 12, at which point a new upper bound is computed having value 50. Nodes 13 and 14 complete the search. The same branch and bound algorithm using the GLB scans 44 nodes of the tree to prove optimality.

4 Implementation

In this section, we outline the branch and bound implementation used in this study. The system is composed of four components: a branch and bound control module; a greedy randomized adaptive search procedure (GRASP) module to produce the initial upper bound; an AMPL modeling language [10] module to manage the linear programming models; and the linear programming solver ADP to produce the lower bounds for the branch and bound algorithm.

The branch and bound control module controls the solution process. It inputs the problem data, calls the GRASP to produce an initial upper bound, and manages the search of the branch and bound tree. At each node of the tree, it spawns an AMPL process that generates the linear program to be solved at that node. The linear program, along with the current best upper bound and the fixed cost associated with the partial assignment of the node, are passed to the LP solver ADP. The optimal permutation is output by the branch and bound module.

The GRASP is called before any search of the branch and bound tree begins so that an initial upper bound can be produced. The GRASP is described in Li, Pardalos, and Resende [20] and its source code is described in Resende, Pardalos, and Li [30]. The branch and bound module passes to the GRASP the problem data and the number of GRASP iterations and gets back the best permutation found over the GRASP iterations and its corresponding cost (the upper bound).

At each node of the branch and bound tree ADP iterates on the LP, producing a sequence of lower bounds, until the dual objective function is greater than the current upper bound minus the fixed cost of the partial assignment, in which case the branch rooted at the current node of the branch and bound tree can be pruned, or if the relative improvement of the dual objective function falls below a specified tolerance, in which case the branch needs to be further

Table 2: Branch and bound algorithm on `nug05`

node	UB	LB	permutation				
1	52	58	1	-	-	-	-
2	52	55	2	-	-	-	-
3	52	52	5	-	-	-	-
4	52	57	3	-	-	-	-
5	52	50	4	-	-	-	-
6	52	57	4	3	-	-	-
7	52	50	4	5	-	-	-
8	52	56	4	5	3	-	-
9	52	56	4	5	2	-	-
10	52	50	4	5	1	-	-
11	52	60	4	5	1	3	-
12	52	50	4	5	1	2	-
	50	-	4	5	1	2	3
13	50	56	4	2	-	-	-
14	50	50	4	1	-	-	-

right children, as follows:

$$\begin{array}{rcl} \text{pick} & i & \notin S_A \\ S_A^l & = & S_A \\ S_E^l & = & S_E \cup \{i\} \\ S_A^r & = & S_A \cup \{i\} \\ S_E^r & = & \emptyset \\ q^l & = & q \\ q^r & = & q \text{ and } q(i) = p(i), \text{where } p \text{ is the incumbent,} \end{array}$$

and the key of left child is the same as the key of the current node and the key of the right child is the newly computed lower bound.

- *Elimination*: The elimination procedure compares the newly computed lower bound of the right child to the incumbent and deletes the right child if its key is greater than the incumbent, thus pruning the entire subtree rooted at the right child.
- *Termination Test*: The algorithm stops if, and only if, the heap is empty.

In the final step, a best permutation found is taken as the global optimal permutation.

- *Elimination:* A newly created subproblem is deleted if its lower bound is greater than or equal to that of the incumbent (the best feasible solution discovered up to that point of the search).

- *Termination Test:* In some cases, with restrictive constraints, it may be possible to define a number of auxiliary rules that help identify infeasible partial solutions.

In the selection procedure, the best-bound and depth-first search strategies are used in most situations. Best-bound search minimizes the number of partial problems decomposed prior to termination. However, it tends to consume an amount of memory that is an exponential function of the problem size. On the other hand, depth-first search consumes an amount of space that is only a linear function of the problem size, and its implementation is relatively easy. The branch-and-bound algorithm terminates when the list of active subproblems is empty, and the incumbent is the optimal solution of the original problem.

The branch and bound algorithm in this paper uses the branch-and-bound technique above. The term solution and permutation are used interchangeably in the discussion. The algorithm consists of three steps.

In the first step, an initial upper bound is computed and an initial branch-and-bound search tree is set up. Our branch and bound tree is a binary tree, each node having a left and right child. For the purpose of describing the branching process, let us denote, at any node of the branch and bound tree, S_A to be the set of already assigned facilities in the partial assignment, S_E the facilities that will never be in the partial assignment in any node of the subtree rooted at the current node. Let S_A^l, S_E^l and S_A^r, S_E^r be the corresponding sets for the left and right children of the current node. Let q denote the partial assignment at the current node. Each node of the branch and bound tree is organized as a heap with a key that is equal to the lower bound on the solution to the original QAP obtainable by any node in the subtree rooted at this node. The binary tree is organized in maximum order, i.e. the node with the largest lower bound is first.

The initial best known upper bound is computed by the GRASP heuristic described in [20, 30]. The initial search tree consists of n nodes with $S_A = \{i\}$ and $S_E = \emptyset$ for $i = 1, \ldots, n$, and $q(i) = p(i)$, where p is the permutation obtained by the GRASP and for $k \neq i$, $q(k) = 0$ and a key of 0.

In the second step, the four procedures of the branch-and-bound as described earlier are:

- *Selection*: The selection procedure simply chooses the partial permutation stored in the root of the heap, i.e. we pick the node with the maximum key.

- *Branching*: The branching procedure creates two children, the left and

The notation of Ibaraki [14] is employed to formally define a branch-and-bound algorithm that will be needed in the sequel. Let P_0 denote an optimization problem and f denote the objective function to be minimized. The decomposition process applied to P_0 is represented by a rooted tree $\mathcal{R} = (\mathcal{P}, \mathcal{E})$, where $\mathcal{P}$ is a set of nodes and $\mathcal{E}$ is a set of arcs. The root of $\mathcal{R}$, denoted P_0, corresponds to the given problem P_0, and other nodes P_i correspond to partial problems P_i. The arc $(P_i, P_j) \in \mathcal{E}$ if, and only if, P_j is generated from P_i by a decomposition. The set of terminal nodes of $\mathcal{R}$, denoted $\mathcal{T}$, are those partial problems that are solved without further decomposition. The level of $P_i \in \mathcal{R}$, denoted $L(P_i)$, is the length of the path from P_0 to P_i in $\mathcal{R}$. P_0 has level 0. $\mathcal{R}$ is assumed to be a finite graph.

A branch-and-bound algorithm attempts to solve P_0 by examining only a small portion of $\mathcal{P}$. This is accomplished by no longer proceeding along the branches rooted at those nodes P_i that are either solved or found by test not to yield an optimal solution of P_0 (i.e., $f(P_i) > f(P_0)$). A lower bound function g is calculated for each subproblem as it is created, to help eliminate unnecessary search. This lower bound function represents a smallest possible cost of a solution to that subproblem, given the subproblem's constraints. Its values satisfy the following conditions

- $g(P_i) \leq f(P_i)$ for $P_i \in \mathcal{P}$,
- $g(P_i) = f(P_i)$ for $P_i \in \mathcal{T}$,
- $g(P_j) \geq g(P_i)$ if P_j is a descendant of P_i.

A typical branch-and-bound algorithm consists of four major procedures: selection, branching, elimination, and termination test.

- *Selection:* At any step during the execution of the algorithm there exists a set $\mathcal{A}$ of problems that have been generated but not yet examined. The selection procedure selects a single subproblem from the set $\mathcal{A}$, based on a selection heuristic function h. The set $\mathcal{A}$ is maintained in an ordered list by increasing values of h. The following three heuristic searching strategies are commonly used:
  - Best-bound search: $h \equiv g$,
  - Depth-first search: $h \equiv -L$, or
  - Breadth-first search: $h \equiv L$, where L is the node level in $\mathcal{R}$.
- *Branching:* A problem specific *branching rule* is used to generate new smaller subproblems from the one selected by the selection procedure. Lower bounds for the newly generated subproblems are calculated accordingly.

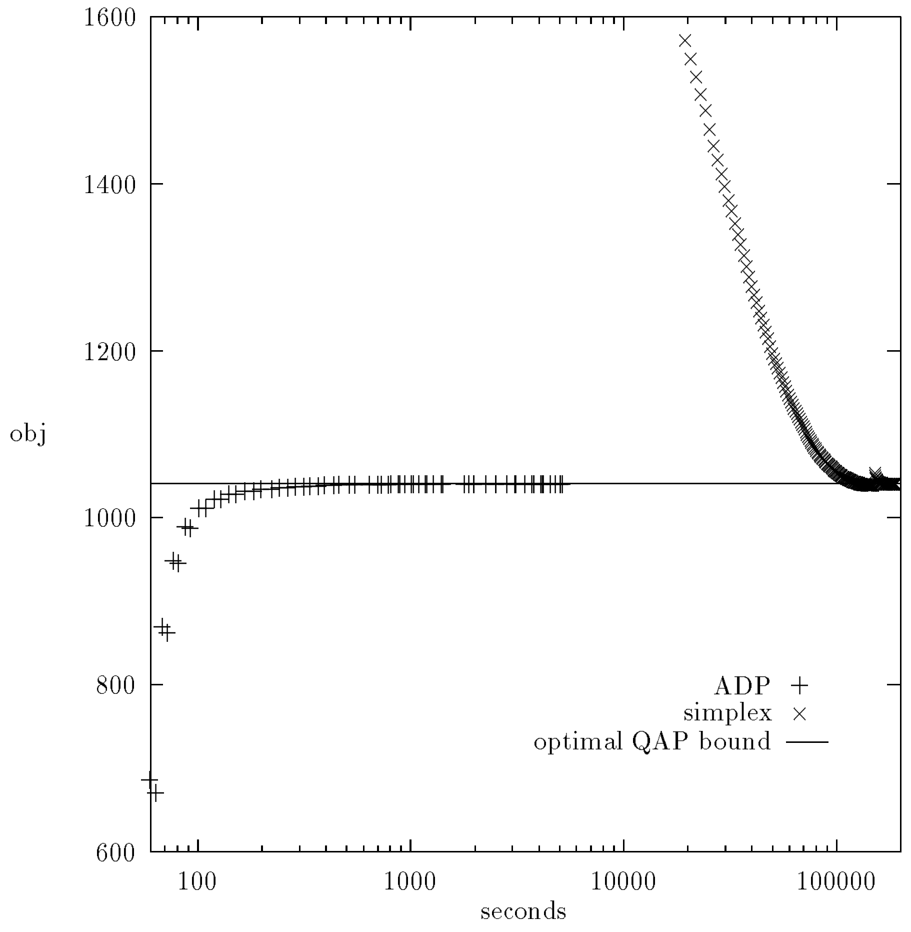

Figure 1: CPLEX simplex and ADP iterates on `nug15`

of QAP test problems. On the majority of the instances tested, the LP-based lower bound was the new best known lower bound. In 87 percent of the cases, the best lower bound was produced. On several instances, including some of dimension $n \geq 20$ the LP-based lower bound is tight.

The linear programs were solved with ADP, an implementation of a dual interior point algorithm, with centering, that uses a preconditioned conjugate gradient algorithm to compute the directions taken at each iteration by the interior point method [15]. Attempts at using the simplex and (direct factorization based) interior point codes of CPLEX [1] were successful only for the smallest instances.

Since the interior point algorithm used solves the dual linear program, it produces a sequence of lower bounds (dual interior solutions), each of which can be compared with the best upper bound to decide if pruning of the search tree can be done at the node on which the lower bound is computed. In the next section, we discuss the branch and bound algorithm in detail. Figure 1 illustrates the sequence of lower bounds produced by ADP, compared to the sequence of feasible primal solutions produced by the primal simplex code of CPLEX on QAPLIB test problem `nug15`. The figure suggests that the algorithm can be stopped many iterations prior to convergence to the optimal value and still be close in value to the optimal solution. This is important in branch and bound codes, where often a lower bound needed to prune the search tree is less than the value of the best lower bound.

3 Branch-and-bound algorithm

Pardalos, Ramakrishnan, Resende, and Li [28] describe a branch and bound algorithm used to study the effectiveness of a variance reduction based lower bound proposed by Li, Pardalos, Ramakrishnan, and Resende [19]. In this paper, we use this branch and bound algorithm in conjunction with the LP-based lower bound described in Section 2. We next describe the branch and bound algorithm and illustrate the algorithm by applying it on a small instance of the QAP.

The underlying idea of a branch-and-bound algorithm is to partition a given initial problem into a number of intermediate partial problems of smaller sizes. Every subproblem is characterized by the inclusion of one or more constraints corresponding to one or more facilities being assigned to specific sites. The decomposition is repeatedly applied to the generated subproblems until each unexamined subproblem is decomposed, solved, or shown not to lead to an optimal solution to the original problem. Branch-and-bound is essentially a variant, or refinement, of backtracking that can take advantage of information about the optimality of partial solutions to avoid considering solutions that cannot be optimal, hence to reduce the search space significantly.

[1]CPLEX is a Registered Trademark of CPLEX Optimization, Inc.

Table 1: Dimension of lower bound linear programs

n	constraints	variables
2	12	6
3	42	27
4	104	88
5	210	225
6	372	486
7	602	931
8	912	1632
9	1314	2673
10	1820	4150
11	2442	6171
12	3192	8856
13	4082	12337
14	5124	16758

subject to:

$$\sum_{j \in I}^{(j>i)} y_{irjs} + \sum_{j \in I}^{(j<i)} y_{jsir} = x_{ir}, \quad i \in I, r \in I, s \in I \; (s \neq r),$$

$$\sum_{s \in I}^{(r \neq s)} y_{irjs} = x_{ir}, \quad i \in I, r \in I, j \in I \; (j > i),$$

$$\sum_{s \in I}^{(r \neq s)} y_{jsir} = x_{ir}, \quad i \in I, r \in I, j \in I \; (j < i),$$

$$\sum_{i \in I} x_{ir} = 1, \quad r \in I,$$

$$\sum_{r \in I} x_{ir} = 1, \quad i \in I,$$

$$0 \leq x_{ir} \leq 1, \quad i \in I, r \in I,$$

$$0 \leq y_{irjs} \leq 1, \quad i \in I, r \in I, j \in I, s \in I,$$

where the set $I = \{1, 2, \ldots, n\}$. This linear program has $n^2(n-1)^2/2 + n^2$ variables and $2n^2(n-1) + 2n$ constraints. Table 1 shows the dimension of theses linear programs for several values of n.

Resende, Ramakrishnan, and Drezner [31] tested the LP-based lower bound on the 63 instances of dimension $n \leq 30$ from QAPLIB [4], a standard library where Π is the set of all permutations of $\{1, 2, \ldots, n\}$, $A = (a_{ij}) \in \mathcal{R}^{n\times n}$, $B = (b_{ij}) \in \mathcal{R}^{n\times n}$. Applications of the QAP are abundant [1, 11, 13, 17, 20, 24, 29]. Many classical combinatorial optimization problems, such as the traveling salesman problem, graph partitioning, and maximum clique are special cases of the QAP.

A wide range of heuristics have been applied to find approximate solutions to the QAP [5, 9, 20, 23, 26, 33, 34, 35]. Exact solution approaches have been limited to instances of dimension $n \le 15$. General QAPs of dimension $n > 15$ are considered difficult large-scale problems.

Branch and bound algorithms have been the most successful methods for proving optimality of quadratic assignment problems. Branch and bound algorithms for the QAP have been studied in [8, 3, 32, 27, 22, 21, 6, 28]. Other exact approaches are described in [29].

Lower bounds are key to the computational efficiency of branch and bound algorithms. Recently, Resende, Ramakrishnanm, and Drezner [31] used an efficient implementation of an interior point algorithm to compute lower bounds for the QAP by solving the linear programming (LP) relaxation of a classical integer programming formulation of the QAP. They showed that the solution time of the LP depended heavily on the algorithm and implementation being used. That study also showed that the quality of the lower bounds produced was good, suggesting that they be incorporated in a branch and bound algorithm. Drezner [7] had previously studied this lower bound.

In this paper, we show how this lower bound performs when implemented in a branch and bound algorithm for the QAP. We report computational experiments on a set of test problems using a branch-and-bound algorithm with the LP-based bounds, as well as a branch-and-bound algorithm that uses the identical branching scheme with the classical Gilmore-Lawler lower bounds (GLB).

The paper is organized as follows. In Section 2 we review the linear programming based lower bounds used in this implementation. The branch and bound algorithm is outlined in Section 3. The implementation is described in Section 4. Computational results are summarized in Section 5 and concluding remarks are made in Section 6.

2 Linear programming lower bounds

Resende, Ramakrishnan, and Drezner [31] consider the following linear program as a lower bound to the optimal solution of a QAP.

$$\min \sum_{i\in I}^{(i<j)} \sum_{r\in I}^{(r\neq s)} \sum_{j\in I} \sum_{s\in I} (a_{ij}b_{rs} + a_{ji}b_{sr})y_{irjs}$$

A Branch and Bound Algorithm for the Quadratic Assignment Problem using a Lower Bound Based on Linear Programming[*]

K.G. Ramakrishnan[†] M.G.C. Resende[‡] P.M. Pardalos[§]

Abstract

In this paper, we study a branch and bound algorithm for the quadratic assignment problem (QAP) that uses a lower bound based on the linear programming (LP) relaxation of a classical integer programming formulation of the QAP. Computational experience with the branch and bound algorithm on several QAP test problems is reported. The linear programming relaxations are solved with an implementation of an interior point algorithm that uses a preconditioned conjugate gradient algorithm to compute directions. The branch and bound algorithm is compared with a similar branch and bound algorithm that uses the Gilmore-Lawler lower bound (GLB) instead of the LP-based bound. The LP-based algorithm examines a small portion of the nodes explored by the GLB-based algorithm.

1 Introduction

The quadratic assignment problem (QAP), first proposed by Koopmans and Beckmann [16], can be stated as

$$\min_{p \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{p(i)p(j)},$$

[*]To appear in Proceedings of *State of the Art in Global Optimization: Computational Methods and Applications*, Princeton University, April 28–30, 1995.

[†]AT&T Bell Laboratories, Murray Hill, NJ 07974 USA. e-mail: `kgr@research.att.com`

[‡]AT&T Bell Laboratories, Murray Hill, NJ 07974 USA. e-mail: `mgcr@research.att.com`

[§]Center for Applied Optimization and Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611 USA. e-mail: `pardalos@ufl.edu`