173.

[41] 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.

[42] P. Pardalos, A. Phillips, and J. Rosen, *Topics in parallel computing in mathematical programming*, Science Press, 1993.

[43] 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.

[44] F. Rendl and H. Wolkowicz, *Applications of parametric programming and eigenvalue maximization to the quadratic assignment problem*, Mathematical Programming, 53 (1992), pp. 63–78.

[45] 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.

[46] 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.

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

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

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

[50] 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.

[13] N. Christofides and M. Gerrard, *A graph theoretic analysis of bounds for the quadratic assignment problem*, in Studies on graphs and discrete programming, P. Hansen, ed., North-Holland, 1981, pp. 61–68.
[14] J. Clausen. Announcement on Discrete Mathematics and Algorithms Network (DIMANET).
[15] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to algorithms*, The MIT Press, 1990.
[16] C. Edwards, *A branch and bound algorithm for the Koopmans-Beckman quadratic assignment problem*, Mathematical Programming Study, 13 (1980), pp. 35–52.
[17] T. Feo and M. Resende, *Greedy randomized adaptive search procedures*, Journal of Global Optimization, 6 (1995), pp. 109–133.
[18] G. Finke, R. Burkard, and F. Rendl, *Quadratic assignment problems*, Annals of Discrete Mathematics, 31 (1987), pp. 61–82.
[19] 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.
[20] R. Francis and J. White, *Facility Layout and Location*, Prentice-Hall, Englewood Cliffs, N.J., 1974.
[21] A. Frieze and J. Yadegar, *On the quadratic assignment problem*, Discrete Applied Mathematics, 5 (1983), pp. 89–98.
[22] P. Gilmore, *Optimal and suboptimal algorithms for the quadratic assignment problem*, J. SIAM, 10 (1962), pp. 305–313.
[23] S. Hadley, F. Rendl, and H. Wolkowicz, *Bounds for the quadratic assignment problem using continuous optimization techniques*, in Integer Programming and Combinatorial Optimization, University of Waterloo Press, 1990, pp. 237–248.
[24] ———, *A new lower bound via projection for the quadratic assignment problem*, Mathematics of Operations Research, 17 (1992), pp. 727–739.
[25] L. Hubert, *Assignment methods in combinatorial data analysis*, Marcel Dekker, Inc., New York, NY 10016, 1987.
[26] T. Ibaraki, *Theoretical comparisons of search strategies in branch-and-bound algorithms*, International Journal of Computer and Information Sciences, 5 (1976), pp. 315–344.
[27] B. Jansen, *A note on "Lower bounds for the QAP"*, tech. rep., Delft University of Technology, Mathematics and Computer Science, December 1993.
[28] T. Kaufman and F. Broeckx, *An algorithm for the quadratic assignment problem using Bender's decomposition*, European Journal of Operational Research, 2 (1978), pp. 204–211.
[29] T. Koopmans and M. Beckmann, *Assignment problems and the location of economic activities*, Econometrica, 25 (1957), pp. 53–76.
[30] J. Krarup and P. Pruzan, *Computer-aided layout design*, Mathematical Programming Study, 9 (1978), pp. 75–94.
[31] E. Lawler, *The quadratic assignment problem*, Management Science, 9 (1963), pp. 586–599.
[32] 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.
[33] 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.
[34] Y. Li and P. M. Pardalos, *Generating quadratic assignment test problems with known optimal permutations*, Computational Optimization and Applications, 1 (1992), pp. 163–184.
[35] 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.
[36] ———, *A new exact algorithm for the solution of quadratic assignment problems*, Discrete Applied Mathematics, (1994). To appear.
[37] H. Mawengkang and B. Murtagh, *Solving nonlinear integer programs with large-scale optimization software*, Annals of Operations Research, 5 (1985/6), pp. 425–437.
[38] E. McCormik, *Human Factors Engineering*, McGraw-Hill, New York, 1970.
[39] 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.
[40] C. Nugent, T. Vollmann, and J. Ruml, *An experimental comparison of techniques for the assignment of facilities to locations*, Journal of Operations Research, 16 (1969), pp. 150–

heuristic found optimal permutation on 23 of the 33 instances solved. This indicates that verification of optimality is the most expensive part of exact algorithms for the QAP.

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 algorithm incorporates a new lower bound based on variance reduction techniques and uses a GRASP heuristic to produce the initial upper bound. The algorithm computes all optimal permutations of the QAP.

The algorithm was compared with an implementation using the Gilmore-Lawler lower bound, and was found to perform better in problems having high data variance in the A and B input matrices. The new algorithm produced optimal solutions for several previously unsolved instances from the QAPLIB.

The data structures incorporated in the branch and bound codes can be useful in other branch and bound approaches for solving quadratic assignment problems. The algorithm can be implemented in parallel to reduce running time requirements [41, 42]. Finally, the branch and bound scheme proposed in this paper, can be implemented with other lower bounds, such the linear programming based lower bounds [1, 46] and lower bounds based on eigenvalues [24, 44].

Acknowledgment. The authors would like to thank the anonymous referees for extensive comments that substantially improved the readability of the paper.

REFERENCES

[1] W. ADAMS AND T. JOHNSON, *Improved linear programming-based lower bounds 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. 43–75.

[2] A. AHO, J. HOPCROFT, AND J. ULLMAN, *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

[3] A. ASSAD AND W. XU, *On lower bounds for a class of quadratic* {0,1} *programs*, Operations Research Letters, 4 (1985), pp. 175–180.

[4] M. BAZARAA AND H. SHERALI, *New approaches for solving the quadratic assignment problem*, Operations Research Verfahren, 32 (1979), pp. 29–46.

[5] D. BERTSEKAS, *Linear network optimization: Algorithms and codes*, MIT Press, Cambridge, MA, 1991.

[6] 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.

[7] R. BURKARD AND T. BONNIGER, *A heuristic for quadratic boolean programs with applications to quadratic assignment problems*, European Journal of Operational Research, 13 (1983), pp. 374–386.

[8] 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.

[9] 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.

[10] R. BURKARD AND F. RENDL, *A thermodynamically motivated simulation procedure for combinatorial optimization problems*, European Journal of Operational Research, 17 (1984), pp. 169–174.

[11] P. CARRARESI AND F. MALUCELLI, *A new lower bound for the quadratic assignment problem*, Operations Research, 40 (1992), pp. S22–S27.

[12] N. CHRISTOFIDES AND E. BENAVENT, *An exact algorithm for the quadratic assignment problem*, Operations Research, 37 (1989), pp. 760–768.

TABLE 5.4
Run statistics – QAPLIB

	GRASP		BB/NLB		BB/GLB	
name	time	up bnd	time	nodes	time	nodes
chr12a	0.45	9916	12.2	36050	0.7	672
chr12b	0.47	9742	3.8	8586	0.6	318
chr12c	0.45	11894	14.7	55845	1.5	3214
chr15a	1.00	11090	594.0	1415685	235.5	413825
chr15b	1.01	9096	93.5	168900	217.8	396255
chr15c	0.98	11366	465.7	1146109	240.0	428722
esc08a	0.07	2	7.3	57464	7.0	57464
esc08b	0.08	8	1.1	6968	0.7	7352
esc08c	0.09	32	0.3	1580	0.3	2552
esc08d	0.08	6	0.3	1448	0.3	2216
esc08e	0.08	2	1.1	10376	1.0	10376
esc08f	0.08	18	0.3	1616	0.3	1520
esc16a	1.01	68	15786.8	58018200	15216.0	60244656
esc16c	1.01	160	133372.4	428754386	-	-
esc16e	0.95	28	18013.8	97558848	14811.1	99030192
esc16g	1.05	26	17844.9	127106352	14542.3	132664368
esc16i	0.98	14	265900.8	1932419536	-	-
lipa10a	0.22	473	0.3	90	0.3	181
lipa10b	0.23	2008	0.3	126	0.3	126
lipa20a	2.56	7506	3182.8	2772772	-	-
lipa20b	2.73	74152	486.0	551	-	-
nug05	0.02	52	0.0	44	0.0	44
nug06	0.04	86	0.1	86	0.1	82
nug07	0.06	148	0.1	127	0.1	115
nug08	0.10	214	0.3	980	0.2	895
nug12	0.40	578	15.7	52626	14.6	49063
nug15	0.97	1152	1012.3	2106172	912.4	1794507
rou10	0.22	174220	0.7	1529	0.8	2683
rou12	0.50	235852	6.5	16309	12.3	37982
rou15	0.96	362518	1276.8	2805138	2240.3	4846805
scr10	0.25	26992	2.9	16162	0.6	1494
scr12	0.46	31410	104.4	408048	4.8	12918
scr15	1.01	51140	2269.3	5609533	274.7	506360

TABLE 5.3
Run statistics – Corner model

	GRASP		BB/NLB		BB/GLB	
name	time	up bnd	time	nodes	time	nodes
rpm-7.1	0.07	209472	0.2	885	0.2	1936
rpm-7.2	0.08	208822	0.2	1228	0.2	2031
rpm-7.3	0.08	212120	0.2	1221	0.2	2133
rpm-7.4	0.08	210140	0.2	1302	0.3	2067
rpm-7.5	0.08	209382	0.2	1372	0.2	2066
rpm-7.6	0.07	211810	0.2	1590	0.2	2010
rpm-7.7	0.07	208334	0.2	1279	0.2	2061
rpm-7.8	0.07	211252	0.2	1175	0.2	2136
rpm-7.9	0.08	210708	0.2	1237	0.2	2094
rpm-7.10	0.07	211808	0.2	1236	0.2	2067
rpm-9.1	0.19	382018	3.1	28288	5.0	54724
rpm-9.2	0.16	379976	3.8	35825	6.7	79991
rpm-9.3	0.18	383808	3.1	27705	6.2	68445
rpm-9.4	0.17	375596	2.8	23856	4.2	49064
rpm-9.5	0.20	383854	3.0	25958	5.0	54282
rpm-9.6	0.17	384638	2.9	28790	4.6	43366
rpm-9.7	0.18	383324	2.3	20965	3.7	41759
rpm-9.8	0.17	379664	3.3	31171	6.3	73613
rpm-9.9	0.17	386990	2.3	20218	3.5	39851
rpm-9.10	0.16	385426	2.5	23205	4.2	48824
rpm-11.1	0.37	635046	636.1	5207959	1206.6	7788836
rpm-11.2	0.36	639678	750.9	5488210	974.3	9895900
rpm-11.3	0.35	638560	471.0	3835941	841.6	8712916
rpm-11.4	0.40	638064	452.0	3649701	828.0	8412364
rpm-11.5	0.40	633000	622.8	5271786	781.7	8018040
rpm-11.6	0.34	628034	502.7	4165664	854.8	8779899
rpm-11.7	0.35	632496	682.6	5789411	941.3	10195879
rpm-11.8	0.37	635824	424.1	3367356	822.9	8327159
rpm-11.9	0.43	618010	400.4	3109865	774.9	7656470
rpm-11.10	0.36	629016	612.7	5070997	943.2	9814496
rpm-13.1	0.66	831154	104141.3	662260712	151791.3	1270116829
rpm-13.2	0.67	821550	100907.4	606049824	117561.9	902560201
rpm-13.3	0.67	844858	44756.9	261310480	83343.2	597555186
rpm-13.4	0.71	826696	94319.0	611865352	133840.2	1164849566
rpm-13.5	0.67	824084	94712.6	609694622	126023.5	1054197091

seed used to generate the instance. The distance and flow matrices are generated as follows. Four squares of size 5×5 are placed in each corner of a 100×100 square and n points are uniformly generated in the small squares such that the number of points in the squares does not differ by more than one. The entries in the distance and flow matrices are the (truncated) Euclidean distances between the points. The instances are available from QAPLIB.

Tables 5.1–5.2 summarize the data characteristics of the problems considered. In each of these tables, we list the problem name, dimension (n), standard deviation (σ) and coefficient of variability (σ/μ) of input matrices A and B, the value of the optimal solution (value) and the number of optimal permutations (perm).

The experiments were conducted on a Silicon Graphics (SGI) Challenge (150 MHz MIPS R4400 processor, 1526 Mbytes of main memory, 16 Kbytes of data cache, and 16 Kbytes of instruction cache). The algorithms were implemented in Fortran and compiled with the `f77` compiler using compiler flags `-O2 -Olimit 800` and times were measured with the system routine `times`.

An upper limit of 2 billion nodes in the search tree was imposed, i.e. all runs that reached 2 billion nodes were terminated. We limit our report to only instances solved within that range. There were no instances for which the algorithm using the Gilmore-Lawler lower bound solved the problem while the one using the new lower bound did not. On the other hand, on several instances, the algorithm with the new lower bound proved optimality of the solution, while the one with the Gilmore-Lawler lower bound scanned the maximum number of search nodes without verifying optimality.

Tables 5.3–5.4 summarize the experimental results on the two sets of test problems. All CPU times are given in seconds. For each instance, the tables list CPU time required by the GRASP and the initial upper bound obtained, and for each of the branch and bound algorithms (BB/NLB = branch and bound algorithms using the new lower bound, BB/GLB = branch and bound algorithm using the Gilmore-Lawler lowr bound), the CPU time and number of search tree nodes processed.

We make the following remarks regarding the experiments.

- On all test problems having high data variance in the A and B matrices, the algorithm with the new lower bound consistently dominated the one with the Gilmore-Lawler lower bound. This becomes more evident with the increase in problem size. In the class of largest problem dimension, the code with the Gilmore-Lawler lower bound processed on average 1.8 times the number of nodes processed by the code with the new lower bound, while taking on average 1.4 times the CPU time. See Table 5.3 for details.
- For the QAPLIB problems, there was a single class having high data variance in both the A and B matrices: `rou`. For that class, the code with the new lower bound also dominated the one using the Gilmore-Lawler lower bound. See Table 5.4 for details.
- Several previously unsolved problems from the QAPLIB were solved to optimality. These were problems `esc16a`, `esc16c`, `esc16e`, `esc16g`, and `esc16i` of dimension $n = 16$, and problems `lipa20a` and `lipa20b` of dimension $n = 20$ [34]. Problems `esc16c`, `esc16i`, `lipa20a`, and `lipa20b` were not solved with the code that uses the Gilmore-Lawler lower bound within the limit of 2 billion search tree nodes.
- For the corner model of test problems, the GRASP heuristic found an optimal permutation on all instances. On the QAPLIB suite of problems, the GRASP

TABLE 5.2
Problem characteristics – QAPLIB

		A		B		optimal sol'n	
name	n	σ	σ/μ	σ	σ/μ	value	perm
chr12a	12	19.702	3.091	28.577	0.634	9552	2
chr12b	12	19.702	3.091	28.577	0.634	9742	1
chr12c	12	19.702	3.091	28.577	0.634	11156	5
chr15a	15	18.437	3.277	31.634	0.699	9896	16
chr15b	15	18.437	3.277	31.634	0.699	7990	5
chr15c	15	18.437	3.277	31.634	0.699	9504	16
esc08a	8	0.294	3.134	0.701	1.122	2	17280
esc08b	8	0.710	1.623	0.701	1.122	8	960
esc08c	8	2.343	1.388	0.701	1.122	32	48
esc08d	8	0.797	1.594	0.701	1.122	6	48
esc08e	8	0.487	2.226	0.701	1.122	2	1344
esc08f	8	1.052	1.295	0.701	1.122	18	96
esc16a	16	0.652	1.704	0.901	0.848	68	13271040
esc16c	16	1.146	1.334	0.901	0.848	160	2064384
esc16e	16	0.526	2.495	0.901	0.848	28	30965760
esc16g	16	0.577	2.546	0.901	0.848	26	46448640
esc16i	16	0.557	2.969	0.901	0.848	14	710277120
lipa10a	10	0.522	0.580	2.753	0.525	473	1
lipa10b	10	3.153	0.713	2.753	0.525	2008	1
lipa20a	20	0.617	0.325	4.498	0.456	7366	22
lipa20b	20	11.788	0.688	4.498	0.456	54152	1
nug05	5	0.891	0.696	1.943	1.104	50	5
nug06	6	0.903	0.650	2.619	1.309	86	4
nug07	7	1.055	0.646	2.418	1.118	148	3
nug08	8	1.098	0.628	3.095	1.286	214	4
nug12	12	1.221	0.571	2.827	1.170	578	4
nug15	15	1.411	0.567	2.817	1.067	1150	6
rou10	10	32.806	0.693	30.696	0.714	174220	1
rou12	12	31.317	0.673	30.301	0.718	235528	3
rou15	15	30.613	0.689	30.263	0.692	354210	6
scr10	10	515.207	2.346	1.214	0.601	26992	1
scr12	12	455.316	2.574	1.221	0.571	31410	8
scr15	15	434.694	2.483	1.331	0.550	51140	2

TABLE 5.1
Problem characteristics – Corner model

		A		B		optimal sol'n	
name	n	σ	σ/μ	σ	σ/μ	value	perm
rpm-7.1	7	47.03	0.64	46.90	0.64	209472	1
rpm-7.2	7	47.07	0.64	46.83	0.64	208822	1
rpm-7.3	7	47.35	0.63	47.05	0.63	212120	1
rpm-7.4	7	46.90	0.64	46.95	0.64	210140	1
rpm-7.5	7	46.57	0.64	47.18	0.64	209382	1
rpm-7.6	7	48.11	0.64	47.25	0.64	211810	2
rpm-7.7	7	47.33	0.64	47.03	0.64	208334	1
rpm-7.8	7	47.70	0.63	46.78	0.63	211252	1
rpm-7.9	7	47.23	0.64	46.99	0.64	210708	1
rpm-7.10	7	47.09	0.64	47.28	0.64	211808	1
rpm-9.1	9	48.70	0.63	48.95	0.63	382018	1
rpm-9.2	9	48.89	0.64	48.82	0.64	379976	1
rpm-9.3	9	49.04	0.63	48.55	0.63	383808	1
rpm-9.4	9	49.37	0.64	48.48	0.64	375596	1
rpm-9.5	9	48.96	0.64	49.15	0.64	383854	1
rpm-9.6	9	48.96	0.64	49.22	0.64	384638	1
rpm-9.7	9	49.05	0.63	48.96	0.63	383324	2
rpm-9.8	9	49.59	0.63	48.13	0.63	379664	1
rpm-9.9	9	49.26	0.64	49.42	0.64	386990	1
rpm-9.10	9	49.54	0.64	49.35	0.64	385426	1
rpm-11.1	11	52.18	0.65	51.21	0.65	635046	1
rpm-11.2	11	52.06	0.66	51.91	0.66	639678	2
rpm-11.3	11	52.17	0.66	51.56	0.66	638560	2
rpm-11.4	11	52.23	0.65	51.67	0.65	638064	1
rpm-11.5	11	51.36	0.66	51.91	0.66	633000	1
rpm-11.6	11	51.60	0.66	51.47	0.66	628034	1
rpm-11.7	11	51.79	0.66	51.92	0.66	632496	1
rpm-11.8	11	52.24	0.66	52.03	0.66	635824	2
rpm-11.9	11	51.70	0.66	50.62	0.66	618010	1
rpm-11.10	11	51.32	0.66	51.96	0.66	629016	1
rpm-13.1	13	51.29	0.68	50.87	0.68	831154	1
rpm-13.2	13	51.11	0.68	50.64	0.68	821550	1
rpm-13.3	13	51.63	0.68	51.69	0.68	844858	2
rpm-13.4	13	51.51	0.68	50.54	0.68	826696	2
rpm-13.5	13	50.94	0.68	51.41	0.68	824084	2

i.e. a binary positional representation, where n is the dimension of the original QAP. With this signature we achieve uniqueness, in the sense that for every pair S_A and S_B there corresponds one, and only one, signature $\sigma(S_A, S_B)$. This is computationally efficient, since it avoids collisions in the hash table. However, note that uniqueness is not necessary.

A *hash table* [2, 15] is a data structure for implementing dictionaries (dynamic sets with the operations of insert, delete, and search). The expected time to search an element in a hash table is $O(1)$, which makes hash tables a computationally effective data structure.

If the signature of this node does not match the signature of any previously examined node, the matrix L of that node is computed and saved in a hash table. Otherwise, the computation of L is unnecessary, since its values can be retrieved from the hash table.

The use of signatures and the hash table, as prescribed above, does not avoid having to solve a linear assignment problem at each node. Nevertheless, it reduces substantially the bulk of the work of computing the entries of L.

The computational effort can further be reduced in case L needs to be computed. Observe that the critical computations are the minimal products. Since we use a constant column reduction scheme, we first determine the partitions $\bar{A}_1$, $\bar{A}_2$ and $\bar{B}_1$, $\bar{B}_2$. The columns of $\bar{A}$, $\bar{B}$, $\bar{A}_2$, $\bar{B}_2$, $\bar{A}_1^\top$, $\bar{B}_1^\top$ need to be sorted. Sorting dominates the computational effort at each step of the minimal product computation. Note that since $\bar{A}_2$ and $\bar{B}_2$ have constant columns, there is no need to sort all of the columns of $\bar{A}_2$ and $\bar{B}_2$. Furthermore, observe that since the columns of $\bar{A}$ and $\bar{B}$ are subvectors of the columns of the original A and B matrices, one can presort the original columns once and store the permutation vectors to be retrieved when the sorted columns of $\bar{A}$ and $\bar{B}$ (of the current node) are needed. We make use of two arrays of pointers for each matrix. Array of type `invf(j)` points to the column number of the original matrix that column j is a sub-vector of. Array of type `preperm(i,k)` is the position of the i-th element in the k-th column in the sorted sequence. Collapsing the retrieved permutation vectors, obviates the need for sorting the columns of $\bar{A}$ and $\bar{B}$. Unfortunately, we still are required to sort the columns of matrices $\bar{A}_1$ and $\bar{B}_1$ at each node. This is done with QuickSort [2]. Thus, the complexity of computing of entries of L is bounded by $\mathcal{O}(n''^2 \log n'')$.

5. Computational Results. In this section, we present experimental results comparing the variance reduction-based branch and bound algorithm with a branch and bound algorithm that differs only in the way the lower bounds are computed. The former algorithm uses the LB2(1.0) variant of the new variance reduction bound, while the latter algorithm uses the Gilmore-Lawler lower bound without reduction. This differs from some other implementations [8, 47] of Gilmore-Lawler based branch and bound algorithms where reductions are carried out. The linear assignment problems that need to be solved to compute both lower bounds are solved with the implementation of the Auction Algorithm [5]. Both algorithms use a GRASP heuristic to compute the initial upper bound. The GRASP was run for 100 iterations on each problem instance.

Two sets of test problems are used in the experiments. The first set is taken from the collection of test problems QAPLIB [9] The second is a new class of test problems, called *corner*, designed to show effectiveness of the new lower bound on problems with high data variance. The instances in this model have names of the form `rpm`-n-m, where n indicated the dimension of the QAP and m is the random

Equivalently, l_{ij} can be written as

$$
\begin{aligned}
(4.2) \qquad l_{ij} \;=\; & \bar{a}^{(1)}_{ii}\bar{b}^{(1)}_{jj} + \bar{a}^{(2)}_{ii}\bar{b}_{jj} + \bar{a}_{ii}\bar{b}^{(2)}_{jj} - \bar{a}^{(2)}_{ii}\bar{b}^{(2)}_{jj} + \\
& \min_{p\in\Pi}\left\{ \sum_{k=1,k\neq i}^{n''} \bar{a}^{(1)}_{ik}\bar{b}^{(1)}_{jp(k)} + \sum_{k=1,k\neq i}^{n''} \bar{a}^{(2)}_{ki}\bar{b}_{p(k)j} + \right. \\
& \left. \sum_{k=1,k\neq i}^{n''} \bar{a}_{ki}\bar{b}^{(2)}_{p(k)j} - \sum_{k=1,k\neq i}^{n''} \bar{a}^{(2)}_{ki}\bar{b}^{(2)}_{p(k)j} \right\}.
\end{aligned}
$$

The minimization problem in 4.2 can be solved by using minimal products [43].

In order to prune the entire branch and bound tree rooted at this particular node, it is desirable to obtain a lower bound on any solution of the original problem with the restriction of fixed q, i.e. any solution obtainable from the branch and bound subtree rooted at the current node. If such a lower bound is available and is larger than the incumbent, then one can fathom the branch and bound subtree rooted at the current node. Let us call such a lower bound $lb(q, S_A, S_B)$. Observe that $lb(q, S_A, S_B)$ is not necessarily a lower bound for the original problem. Since a partial assignment q exists at this node, it can be advantageously combined with the lower bound available for the reduced problem (Theorem 3.1) to obtain $lb(q, S_A, S_B)$.

$$
(4.3) \qquad l'_{ij} = l_{ij} + \sum_{k\in S_A} a_{ik}b_{j\,q(k)} + \sum_{k\in S_A} a_{ki}b_{q(k)j}.
$$

Let lb^* be the optimal solution to the linear assignment problem with costs l'_{ij}. $lb(q, S_A, S_B)$ is defined by

$$
(4.4) \qquad lb(q, S_A, S_B) = lb^* + \sum_{k\in S_A} a_{km}b_{q(k)q(m)} + a_{mk}b_{q(m)q(k)}.
$$

THEOREM 4.1. *$lb(q, S_A, S_B)$ is a lower bound on any solution obtained from the nodes of the branch and bound subtree rooted at the current node.*

The proof of this theorem follows along the same lines of the proof of Theorem 3.1 given in [32].

The following lemmata, whose proofs follow from the above discussion, characterize the properties of $lb(q, S_A, S_B)$ that are useful in the implementation.

LEMMA 4.1. *A node of the branch and bound tree is uniquely determined by its descriptor, the tuple (q, S_A, S_B).*

LEMMA 4.2. *The matrix L of the reduced subproblem is uniquely determined by S_A and S_B, i.e. two branch and bound nodes having the same S_A and S_B will have identical matrix L.*

LEMMA 4.3. *In the complete branch and bound tree, there are $n'!$ nodes whose descriptor has identical index sets S_A and S_B.*

Note that, for all $n'!$ branch and bound nodes, the values L are identical. The implementation of the branch and bound algorithm exploits this key property. To do this, we first need a definition. Define the *signature* of a node in a branch and bound tree to be a function of S_A, and S_B of that corresponding node. As the branch and bound tree is traversed, the signature of each node is computed. In our implementation the signature is given by

$$
\sigma(S_A, S_B) = 2^n \cdot \sum_{i\in S_A} 2^{i-1} + \sum_{j\in S_B} 2^{j-1},
$$

is greater than the incumbent, thus pruning the entire subtree rooted at the left 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.

The binary search tree has many interesting properties. First, observe that S_E^l is set to $\emptyset$. This is a consequence of the relationship between S_A and S_E at every node of the branch and bound tree (as enumerated below). The S_A^l implicitly captures the excluded assignment in S_E^l and so S_E^l can be set to an $\emptyset$. Other interesting properties are listed below. All these properties enable us to derive the result on the maximum depth of the branch and bound tree, and the maximum number of nodes in the branch and bound tree.

We denote by L the level of the binary tree, counting the root of the branch and bound tree as level 1. The following properties hold for the branch and bound tree:

- For any node of the branch and bound tree, if $S_E \neq \emptyset$, then all assignments in S_E have exactly one facility index and that index is one larger than the largest facility index in S_A.
- All site indices in $S_A \cup S_E$ are distinct.
- For any node, $|S_A| + |S_E| \leq n$.
- A node i is a right-ancestor of node j if node i is in the path from node j to the root of the branch and bound tree, and i is the right child of its parent. This definition considers node i to be a right-ancestor of itself ($i = j$ in the definition) if i is a right child. Let r_i be the number of right-ancestors for node i. At any level L of the branch and bound tree, and for any node i the following relation holds:

$$|S_A| + r_i = L.$$

- For any node i of the branch and bound tree, we have that $r_i \leq n^2$.
- The maximum depth of the branch and bound tree is n^2. This property gives a bound of at most 2^{n^2} branch and bound nodes.

4. Efficient Computation of the New Lower Bound. To implement the new lower bound in the above branch and bound scheme, we exploit some properties of the bound. At each node of the branch and bound tree, the matrix L must be computed and the corresponding linear assignment problem solved.

At a particular node of the branch and bound tree, let n' denote the number of facilities already assigned to sites. Let q be the corresponding partial assignment vector. Let S_A and S_B denote the index sets of already assigned facilities and sites, respectively (corresponding to the partial assignment q). Note that $|S_A| = |S_B| = n'$. At the current node, it remains to be solved a quadratic assignment problem of reduced size $n - n'$. Theorem 3.1 can be used to obtain a lower bound for the reduced problem.

Let $n'' = n - n'$ be the size of the reduced problem, and let $\bar{A}$ and $\bar{B}$ be the corresponding flow and distance matrices for the the reduced problem. Recall from the theorem, that for $i, j = 1, \ldots, n''$, the element l_{ij} of L is given by

$$l_{ij} = \min_{p \in \Pi, p(i)=j} \sum_{k=1}^{n''} \bar{a}_{ik}^{(1)} \bar{b}_{jp(k)}^{(1)} + \sum_{k=1}^{n''} \bar{a}_{ki}^{(2)} \bar{b}_{p(k)j} + \sum_{k=1}^{n''} \bar{a}_{ki} \bar{b}_{p(k)j}^{(2)} - \sum_{k=1}^{n''} \bar{a}_{ki}^{(2)} \bar{b}_{p(k)j}^{(2)}. \tag{4.1}$$

Note that, as $\theta \to 1$, $\delta_{ij}(\theta) \to \gamma(A) - a_{ij}$, which is the constant column reduction partitioning scheme. Experimentally, we have observed that the constant column reduction partitioning scheme is more effective and is easier to implement than the closed form solution.

3.2. The New Branch-and-Bound Algorithm for QAP. The exact algorithm presented in this section uses the branch-and-bound technique described in Section 2. 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 forest of n binary trees (not necessarily a complete binary tree). Each node of the tree has a left and a 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 assignments (of facilities to sites) that are always fixed at any node of the subtree rooted at this node (including this node), and S_E to be the set of excluded assignments, i.e. the assignments that are forever excluded in any node of the subtree rooted at the current node (including the current node). The sets S_A and S_E completely describe a node of the branch and bound tree. 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. Currently unexplored nodes of the branch and bound tree are 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 heap 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 [33, 45]. Let $P = (p_1, p_2, \ldots, p_n)$ denote the initial solution found by the GRASP heuristic; i.e. p_i is the site assigned to facility i in this solution. We use the notation $\{i \to s\}$ to indicate that facility i is assigned to site s. The initial search tree consists of a forest of n isolated nodes, where, for $i = 1, \ldots, n$, S_A of node i is $\{1 \to p_i\}$, $S_E = \emptyset$, and all n nodes have a key of 0.

In the second step, the four procedures of the branch-and-bound algorithm, as described in Section 2, are used as follows:

- *Selection*: The selection procedure simply chooses the node at the root of the heap, i.e. the node with the maximum key.
- *Branching*: The branching procedure creates two children, the left and the right children, as follows: Let i be the smallest index of a facility that is not in any assignment of S_A, and s be the index of a site that is not in any assignment of S_A, such that the assignment $\{i \to s\}$ is not in S_E. Then,

$$\begin{aligned} S_A^l &= S_A \cup \{i \to s\}, \\ S_E^l &= \emptyset, \\ S_A^r &= S_A, \\ S_E^r &= S_E \cup \{i \to s\}, \end{aligned}$$

  and the key of right child is the same as the key of the current node and the key of the left child is the newly computed lower bound.
- *Elimination*: The elimination procedure compares the newly computed lower bound of the left child to the incumbent and deletes the left child if its key

$$V(M) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\gamma(M)-m_{ij})^2,$$

$$T(M,\lambda) = \lambda\sum_{i=1}^{n}V(m_i)+(1-\lambda)V(M),\ \text{for } 0\le\lambda\le 1.$$

In our reduction scheme, we considered the partition $A = A_1 + A_2$, where $A_1 = A+\Delta$ and $A_2 = -\Delta$, such that the variances of A_1 and A_2, the sum of variances of the rows of A_1, and the sum of variances of the rows of A_2 are minimized. This minimization problem has been formulated as

$$\min \quad \theta\, T(A+\Delta,\lambda)+(1-\theta)\,T(-\Delta^{\top},\lambda) \tag{3.2}$$

$$\text{such that} \quad \Delta\in R^{n\times n} \tag{3.3}$$

where λ and θ $(0\le\lambda,\theta\le 1)$ are input parameters.

Motivated by the observation that for QAP(A,B) the tightness of GLB is inversely proportional to the variances of A and B, the following reduction schemes were proposed in [32]:

- $\mathcal{R}$-1: $a_{ij}^{(1)} = a_{ij}-\theta(a_{nn}-a_{ij})$ and $a_{ij}^{(2)} = \theta(a_{nn}-a_{ij})$, $i,j=1,...,n$,
- $\mathcal{R}$-2: $a_{ij}^{(1)} = a_{ij}-\theta(\gamma(a_n^T)-\gamma(a_j^T))$ and $a_{ij}^{(2)} = \theta(\gamma(a_n^T)-\gamma(a_j^T))$, $i,j=1,...,n$.

Note that both reduction schemes are independent of the value of λ (c.f. [32]). One new lower bound proposed in [32] is to use reduction scheme $\mathcal{R}$-1. This lower bound is denoted by LB1(θ). The other new lower bound proposed is to use reduction scheme $\mathcal{R}$-2. This lower bound is denoted LB2(θ). Both new lower bounds depend on the parameter θ. Note that, LB1(0.0) = GLB(A,B) and LB1(1.0) = GLB(A^T,B^T).

In [32], it was observed empirically that $\theta = 0.5$ and $\theta = 1.0$ are good choices for LB1(θ) and LB2(θ), respectively. Furthermore, in that study the bound LB2(1.0) was slightly tighter than LB1(0.5). Consequently, in this implementation we use LB2(1.0).

For LB2(1.0) the computation is simpler. The variance minimization problem (3.2 –3.3) becomes

$$\min \quad V(A+\Delta) \tag{3.4}$$

$$\text{such that} \quad \Delta\in R^{n\times n}. \tag{3.5}$$

The solution of (3.4 –3.5) is simply

$$\delta_{ij} = \gamma(a_j^{\top})-\gamma(a_n^{\top}),\ \text{for } i,j=1,\ldots,n,$$

which is the constant column partitioning scheme.

The new lower bounds can be computed efficiently. Computing matrix Δ to partition matrices A and B takes only $O(n^2)$ time (c.f. [32]). By presorting the rows of the flow and distance matrices A and B, one can compute l_{ij} $(i,j=1,...,n)$ in $O(n^3)$. Hence the total running time is $O(n^3)$, which is the same as that for computing GLB. Furthermore, the constant factor is small. Later in this paper, we show how to efficiently incorporate these bounds into a branch and bound algorithm for the QAP.

Recently, Jansen [27] has derived an analytical closed form solution to (3.2–3.3). That solution is given by

$$\delta_{ij}(\theta) = \theta\lambda\frac{1-\theta}{1-\theta\lambda}\gamma(a_i)+\frac{\theta(1-\lambda)+\theta\lambda^2(1-\theta)-\theta^2\lambda^2(1-\theta)}{(1-\theta\lambda)(1-\lambda+\theta\lambda)}\gamma(A)$$
$$-\frac{\lambda\theta(1-\theta)}{1-\lambda+\theta\lambda}\gamma(a_j^{\top})-\theta a_{ij}.$$

Fortran source code for solving exactly quadratic assignment problems with a branch and bound algorithm is listed. Roucairol [47] proposed and implemented sequential and parallel branch and bound algorithms on a Cray X-MP/48 (four processors) and solved the Nugent-12 ($n = 12$) test problem [40] in about 5 minutes, but was unable to solve the Nugent-15 ($n = 15$) instance, due to insufficient memory. Pardalos and Crouse [41] developed another parallel implementation of a single assignment branch and bound algorithm on an IBM 3090/400E (four processors). That implementation solved the Nugent-12 problem in half a minute and partially solved (examined 95% of the nodes of the branch and bound tree) in about 30 minutes. More recently, Mautor and Roucairol [36, 35] considered new approaches to reduce the size of the search tree in an exact branch and bound algorithm, and report computational results for some problems of size up to $n = 20$. The long standing problem Nugent-20 ($n = 20$) was reportedly solved in 1994 by Clausen [14]. In addition, QAPs of size up to $n = 30$, in which the flow matrix is the weighted adjacency matrix of a tree, have been solved exactly, using dynamic programming approaches [12]. Other exact approaches are described in [43].

In this paper, we discuss an exact branch and bound algorithm, that incorporates the new lower bound using efficient data structure techniques and the GRASP heuristic [17] to find the initial upper bound.

3.1. A New Class of Lower Bounds. A class of lower bounds based on optimal reduction schemes for the QAP was proposed in [32]. For a given QAP(A, B), consider a partition of A into two matrices $A_1 = (a_{ij}^{(1)})$ and $A_2 = (a_{ij}^{(2)})$ such that $A = A_1 + A_2$ and a partition of B into two matrices $B_1 = (b_{ij}^{(1)})$ and $B_2 = (b_{ij}^{(2)})$ such that $B = B_1 + B_2$. For each pair $\{i, j\}$, $i, j = 1, ..., n$, let

$$l_{ij} = \min_{p \in \pi,\ p(i)=p(j)} \left\{ \sum_{k=1}^{n} a_{ik}^{(1)} b_{jp(k)}^{(1)} + \sum_{k=1}^{n} a_{ki}^{(2)} b_{p(k)j} + \sum_{k=1}^{n} a_{ki} b_{p(k)j}^{(2)} - \sum_{k=1}^{n} a_{ki}^{(2)} b_{p(k)j}^{(2)} \right\} \tag{3.1}$$

where π is the set of all permutations of $\{1, 2, \ldots, n\}$. Let $L = (l_{ij})$ be an $n \times n$ matrix. The following theorem defines a new lower bound [32, Theorem 4.1].

THEOREM 3.1. *Let the matrix L be defined as above. The solution of the linear assignment problem with cost matrix L is a lower bound for the corresponding QAP.*

The classical Gilmore-Lawler bound [22, 31] (denoted here by GLB(A,B)) is a special case in which neither matrix A nor B are partitioned. Different ways of partitioning the matrices A and B (also referred to as *reduction*) yield different lower bounds. The common reduction techniques used in the literature choose A_2 and B_2 with constant column sums (often called constant columns). We refer to such techniques as constant column reductions.

Let $M = (m_{ij})$ be a matrix in $R^{n \times n}$. We treat a row vector m_i, $1 \le i \le n$, of M as a $1 \times n$ matrix and a column vector m_j^T, $1 \le j \le n$, as a $n \times 1$ matrix. For convenience of discussion, we use the following notation for average $\gamma(M)$, variance $V(M)$, and total variance $T(M, \lambda)$ of M:

$$\gamma(M) = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} m_{ij},$$

- $g(P_j) \geq g(P_i)$ 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 *branching rule* related to a given problem 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.
- *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.

3. Branch-and-Bound Algorithms for the QAP. Three classes of methods have been used to find globally optimal solutions to the quadratic assignment problem. These methods include cutting plane techniques, branch and bound methods and dynamic programming.

Exact cutting plane algorithms have not succeeded to generate optimal solutions for problems with dimension as small as $n = 10$ [4, 28]. They have, however, been successfully applied to obtain good suboptimal permutations [7].

Branch and bound algorithms have been the most successful methods for proving optimality of quadratic assignment problems. Lower bounds are key to the computational performance of these branch and bound algorithms. Lower bounds for the quadratic assignment problem can be categorized into three groups. The first category includes the classical Gilmore-Lawler bound (GLB) [22, 31] and related bounds. The second category consists of eigenvalue-based bounds [18, 24, 23, 44]. The rest of the bounds are mostly based on reformulations of the QAP and generally involve solving a number of linear assignment problems (e.g. [3, 11, 13, 16, 21]). A new class of lower bounds, that belongs to the first category, was proposed by Li, Pardalos, Ramakrishnan and Resende [32], and is described in subsection 3.1.

One of the first exact branch and bound algorithms for the QAP is described in [16], but no computational results are reported. In the book by Burkard and Derigs [8],

Branch-and-bound is an enumerative technique for solving combinatorial optimization problems. Branching usually refers to a successive partitioning of the feasible domain while bounding refers to the determination of lower and upper bounds for the global optimal solution. Recently, Li, Pardalos, Ramakrishnan, and Resende [32] proposed new lower bounds, based on reduction techniques, for the QAP. In this paper, we show how to efficiently implement these bounds in a branch and bound algorithm for the QAP. We report on computational experiments with a branch-and-bound algorithm using the new bounds, as well as the Gilmore-Lawler lower bounds, on a large set of test problems.

Before we conclude the introduction, let us define some notation and state some assumptions used in this paper. Matrix A is referred to as the *flow matrix* and B as the *distance matrix*. For convenience of discussion, an instance of the QAP with flow and distance matrices A and B is denoted as QAP(A, B). Without loss of generality, it is assumed that the entries of matrices A and B are nonnegative [43]. We further assume that the diagonal entries of matrices A and B are zero.

The paper is organized as follows. In Section 2 we discuss issues related to branch and bound algorithms. A specialized branch and bound approach for the QAP is given in Section 3. In Section 4, an efficient implementation of the branch and bound algorithm is considered. Computational results are summarized in Section 5 and concluding remarks are made in Section 6.

2. Branch-and-Bound Algorithms. 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 constraint. 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.

The notation of Ibaraki [26] 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 $\mathcal{R}$. 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., $F(P_i) > F(P_0)$, where $F(P) = \min_{x \in P} f(x)$). 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 $P_i \in \mathcal{T}$,

IMPLEMENTATION OF A VARIANCE REDUCTION BASED LOWER BOUND IN A BRANCH AND BOUND ALGORITHM FOR THE QUADRATIC ASSIGNMENT PROBLEM*

P.M. PARDALOS†, K.G. RAMAKRISHNAN‡, M.G.C. RESENDE§ AND Y. LI¶

Abstract. The efficient implementation of a branch and bound algorithm for the quadratic assignment problem (QAP), incorporating the lower bound, based on variance reduction, of Li, Pardalos, Ramakrishnan, and Resende (1994), is presented. A new data structure for efficient implementation of branch and bound algorithms for the QAP is introduced. Computational experiments with the branch and bound algorithm on different classes of QAP test problems are reported. The branch and bound algorithm using the new lower bounds is compared with the same algorithm utilizing the commonly applied Gilmore-Lawler lower bound. Both implementations use a greedy randomized adaptive search procedure for obtaining initial upper bounds. The algorithms report all optimal permutations. Optimal solutions for previously unsolved instances from the literature, of dimensions $n = 16$ and $n = 20$, have been found with the new algorithm. In addition, the new algorithm has been tested on a class of large data variance problems, requiring the examination of much fewer nodes of the branch and bound tree than the same algorithm using the Gilmore-Lawler lower bound.

Key Words. Combinatorial optimization, quadratic assignment problem, branch and bound, GRASP, computer implementation, data structures, hashing function, hash table, lower bound, test problems

AMS(MOS) subject classification. 90B80, 90C20, 90C35, 90C27, 65H20, 65K05

1. Introduction.

The quadratic assignment problem (QAP) can be stated as

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

where Π is the set of all permutations of $\{1, 2, \ldots, n\}$, $A = (a_{ij}) \in \mathcal{R}^{n \times n}$, $B = (b_{ij}) \in \mathcal{R}^{n \times n}$. The QAP was first proposed by Koopmans and Beckmann in 1957 as a mathematical model for a set of indivisible economical activities [29]. A typical example of the QAP is the facility location problem, in which a set of n facilities is to be assigned to an equal number of locations. Between each pair of facilities, there is a given amount of flow, contributing a cost equal to the product of the flow and the distance between the locations to which the facilities are assigned. Applications of the QAP are abundant, and can be found in [6, 20, 25, 30, 33, 38, 43]. Many classical combinatorial combinatorial optimization problems, such as the traveling salesman problem and the graph partitioning problem, are special cases of the QAP.

A wide range of heuristics have been applied to find approximate solutions to the QAP [10, 19, 33, 37, 39, 48, 49, 50]. Exact solution approaches have been limited to instances of dimension $n \leq 15$, and are most based on branch and bound. General quadratic assignment problems of dimension $n > 15$ are considered difficult large-scale problems.

* September 6, 1995 – To appear in *SIAM Journal on Optimization*

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

‡ 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

¶ Sanwa Financial Products Company, 55 East 52nd Street, 26th Floor, New York, NY 10055 USA. e-mail: yong@sanwaBGK.com