Computing Lower Bounds for the Quadratic Assignment Problem with an Interior Point Algorithm for Linear Programming*

M.G.C. Resende† K.G. Ramakrishnan‡ Z. Drezner§

Abstract

A typical example of the quadratic assignment problem (QAP) is the facility location problem, in which a set of n facilities are to be assigned, at minimum cost, 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 locations to which the facilities are assigned. Proving optimality of solutions to quadratic assignment problems has been limited to instances of small dimension (n less than or equal to 20), in part because known lower bounds for the QAP are of poor quality. In this paper, we compute lower bounds for a wide range of quadratic assignment problems using a linear programming-based lower bound studied by Drezner (1994). On the majority of quadratic assignment problems tested, the computed lower bound is the new best known lower bound. In 87 percent of the instances, we produced the best known lower bound. On several instances, including some of dimension n equal to 20, the lower bound is tight. The linear programs, which can be large even for moderate values of n, are solved with an interior point code that uses a preconditioned conjugate gradient algorithm to compute the directions taken at each iteration by the interior point algorithm. Attempts to solve these instances using the CPLEX primal simplex algorithm as well as the CPLEX barrier (primal-dual interior point) method were successful only for the smallest instances.

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 proposed by Koopmans and Beckmann (1957) as a mathematical model for a set of indivisible economical activities. 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 (Bokhari, 1987; Francis & White, 1974; Hubert, 1987; Krarup

*Revision date: March 21, 1995

†AT&T Bell Laboratories, Murray Hill, NJ 07974 USA. E-mail: mgcr@research.att.com

‡AT&T Bell Laboratories, Murray Hill, NJ 07974 USA. E-mail: kgr@research.att.com

§Department of Management Science, School of Business Administration and Economics, California State University, Fullerton, CA 92634 USA. E-mail: drezner@fullerton.edu

& Pruzan, 1978; Li, Pardalos, & Resende, 1994b; McCormick, 1970; Pardalos & Wolkowicz, 1994). Many classical 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 (Burkard & Rendl, 1984; Fleurent & Ferland, 1994; Li et al., 1994b; Mawengkang & Murtagh, 9856; Murtagh, Jefferson, & Sornprasit, 1982; Skorin-Kapov, 1990; Taillard, 1991; Thonemann, U.W. and Bölte, A.M., 1994). Exact solution approaches have been limited to small ($n \leq 20$) instances, and are mostly based on branch and bound. This has been frequently attributed to the observation that lower bounds for the QAP tend to deteriorate quickly, as the size of the QAP increases.

Lower bounds can be categorized into three groups. The first includes the Gilmore-Lawler lower bound (Gilmore, 1962; Lawler, 1963; Li, Pardalos, Ramakrishnan, & Resende, 1994a) and related bounds. Eigenvalue-based bounds (Finke, Burkard, & Rendl, 1987; Hadley, Rendl, & Wolkowicz, 1990, 1992a, 1992b) constitute the second category. They have been generally acknowledged to be the best, but also the most expensive to compute. The third group of bounds are mainly based on reformulations of the QAP (Assad & Xu, 1985; Carraresi & Malucelli, 1992; Chakrapani & Skorin-Kapov, 1994; Christofides & Gerrard, 1981; Frieze & Yadegar, 1983; Karisch & Rendl, 1994) and are usually computed by solving a series of linear assignment problems.

Drezner (1994) studied a lower bound based on the linear programming (LP) relaxation of a well-known integer programming formulation of the QAP. Computational testing was done on three small quadratic assignment problems, showing that on those instances the LP relaxations produced tight bounds. In fact, they also produce an optimal permutation. In this paper, we extend Drezner's experiments to a large set of QAP test problems from QAPLIB (Burkard, Karisch, & Rendl, 1991), a much-studied suite of problems.

For a QAP of dimension n, the LP-based bound requires the solution of a linear program having $2n^2(n-1)+2n$ constraints and $n^2(n-1)^2/2+n^2$ variables. For even moderate values of n, the resulting linear programs are large, by today's standards. For example, to compute the LP-based bound for a quadratic assignment problem of dimension $n = 30$, requires the solution of a linear program with 52,260 constraints and 379,350 variables. To solve linear programs of up to this size, we apply an interior point LP code (Karmarkar & Ramakrishnan, 1991) that uses a conjugate gradient algorithm to approximately compute the improving direction at each iteration of the interior point algorithm. Attempts to solve these linear programs with the commercially available CPLEX (CPLEX is a Registered Trademark of CPLEX Optimization, Inc.) primal simplex and CPLEX barrier (primal-dual) interior point algorithms indicate that, except for the smaller instances, these linear programs are beyond the current capabilities of those solvers. CPLEX version 2.1 was used in the experiments reported in this paper.

The paper is organized as follows. In Section 1 we review the LP-based bound. In Section 2, we outline how we compute the bounds, and briefly discuss our attempts at solving the LP relaxations using CPLEX. Computational results are summarized in Section 3 and concluding remarks are made in Section 4.

1 A Linear Programming Bound for QAP

Define the $(0,1)$-variable x_{ir} to be such that $x_{ir} = 1$ if, and only if, facility i is assigned to location r, and let c_{ij}^{rs} be the cost associated with simultaneously assigning facility i to location r and facility j to location s. The following is a classical integer quadratic programming formulation for the quadratic assignment problem:

$$\min \qquad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{r=1}^{n}\sum_{s=1}^{n} c_{ij}^{rs} x_{ir} x_{js} \tag{1}$$

subject to:

$$\begin{aligned}
\sum_{i=1}^{n} x_{ir} &= 1, \quad r = 1, \ldots, n, && (2) \\
\sum_{r=1}^{n} x_{ir} &= 1, \quad i = 1, \ldots, n, && (3) \\
x_{ir} &= \{0,1\}, \quad i, r = 1, \ldots, n. && (4)
\end{aligned}$$

By defining the $(0,1)$-variable $y_{irjs} = x_{ir}x_{js}$, it follows that, for $i, r, s = 1, \ldots, n$,

$$\begin{aligned}
\sum_{j=1}^{n} y_{irjs} &= \sum_{j=1}^{n} x_{ir}x_{js} \\
&= x_{ir} \sum_{j=1}^{n} x_{js} \\
&= x_{ir},
\end{aligned}$$

and for $i, j, r = 1, \ldots, n$,

$$\begin{aligned}
\sum_{s=1}^{n} y_{irjs} &= \sum_{s=1}^{n} x_{ir}x_{js} \\
&= x_{ir} \sum_{s=1}^{n} x_{js} \\
&= x_{ir}.
\end{aligned}$$

Substituting y_{irjs} into (1–4) gives us the following integer programming formulation for the QAP:

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{r=1}^{n}\sum_{s=1}^{n} c_{ij}^{rs} y_{irjs} \tag{5}$$

subject to:

$$\begin{aligned}
\sum_{i=1}^{n} x_{ir} &= 1, \quad r = 1, \ldots, n, && (6) \\
\sum_{r=1}^{n} x_{ir} &= 1, \quad i = 1, \ldots, n, && (7) \\
\sum_{j=1}^{n} y_{irjs} &= x_{ir}, \quad i, r, s = 1, \ldots, n, && (8) \\
\sum_{s=1}^{n} y_{irjs} &= x_{ir}, \quad i, j, r = 1, \ldots, n, && (9) \\
x_{ir} &= \{0,1\}, \quad i, r = 1, \ldots, n, && (10) \\
y_{irjs} &= \{0,1\}, \quad i, r, j, s = 1, \ldots, n. && (11)
\end{aligned}$$

Note that for $i, j, r, s = 1, \ldots, n$, we have $y_{irjs} = x_{ir}x_{js} = x_{js}x_{ir} = y_{jsir}$, implying

$$y_{irjs} = y_{jsir}. \tag{12}$$

Consequently, (12) can be incorporated into the objective function and constraints of the integer program (5–11), reducing substantially the number of variables and constraints. Furthermore, $y_{irjs} = 0$ if $i = j$ $(r \neq s)$ or $r = s$ $(i \neq j)$, and can be thus eliminated from the formulation for those indices.

Table 1: The Nugent *et al.* test problems

name	n	LP relaxation constraints	variables	nz	bks	glb
nug05	5	210	225	1050	50	50
nug06	6	372	486	2232	86	82
nug07	7	602	931	4214	148	137
nug08	8	912	1632	7296	214	186
nug12	12	3192	8856	38304	578	493
nug15	15	6330	22275	94950	1150	963
nug20	20	15240	72600	304800	2570	2057
nug30	30	52260	379350	1567800	6124	4539

This results in an integer linear program with $n^2(n-1)^2/2 + n^2$ variables and $2n^2(n-1) + 2n$ constraints.

Drezner (1994) proves that the optimal objective function value of the linear programming relaxation of the integer program (5–11) obtained by relaxing the integrality requirements of (10) and (11) with the linear constraints

$$x_{ir} \geq 0, \quad i, r = 1, \ldots, n, \tag{13}$$

$$y_{irjs} \geq 0, \quad i, r, j, s = 1, \ldots, n, \tag{14}$$

is a lower bound for the QAP that is at least as good as the classical Gilmore-Lawler lower bound. Adams and Johnson (1994) show that the Gilmore-Lawler lower bound is equivalent to the above linear programming formulation with (12) removed and develop a dual-ascent procedure for approximating the dual of the LP relaxation. In that paper, however, they take at most 50 iterations of the dual-ascent procedure, failing to reach the optmal value on all instances but the smallest (of dimension $n = 5$). It is not clear whether the lower bound would improve if additional dual-ascent iterations were to be taken.

Let $c^\top y^*$ be the optimal solution to the LP relaxation (5–10, 13–14). Assuming integer data in the QAP matrices, we call $\lceil c^\top y^* \rceil$ the QAPLP lower bound, where $\lceil z \rceil$ is the smallest integer greater than or equal to z. If the QAP matrices are symmetric, the LP objective function value can be further rounded to the smallest even number greater than or equal to it. Throughout this paper, we shall often refer to the best known approximate solution to the dual linear program, rounded up, also as the QAPLP bound.

2 Efficient Computation of the LP-Based Lower Bound

QAPLIB (Burkard et al., 1991) is a collection of QAP test instances commonly used to test codes that compute lower bounds, optimal or approximate solutions to quadratic assignment problems. Perhaps one of the most used classes of QAP test problems is the one introduced by Nugent, Vollmann, and Ruml (Nugent, Vollmann, & Ruml, 1969). QAPLIB has eight instances from this class, summarized in Table 1, where for each instance, the dimension, number of constraints, variables, and nonzero elements in the constraint matrix of the LP relaxation, best known solution (bks), and Gilmore-Lawler lower bound (glb) are listed. Drezner (Drezner, 1994) solved the LP relaxations of instances `nug05`, `nug06`, and `nug07`, finding tight bounds (i.e. 50, 86, and 148, respectively) for each of the three instances. To expand on the experimental results of Drezner, we attempted to solve the LP relaxations with the commercially available LP solver CPLEX. We used two algorithms in CPLEX Version 2.1: the default CPLEX primal simplex algorithm and the CPLEX barrier (interior point) algorithm. Table 2 summarizes results for those runs on a Silicon Graphics Challenge computer

Table 2: CPLEX 2.1 on Nugent *et al.* test problems

	simplex			barrier			
name	sol'n	itr	time	sol'n	b-itr	x-time	time
nug05	50.00	103	0.23	50.00	8	0.12	0.58
nug06	86.00	551	2.25	86.00	8	0.28	3.37
nug07	148.00	2813	21.95	148.00	11	1.10	17.66
nug08	203.50	5960	91.26	203.50	10	1.79	57.22
nug12	522.89	57524	9959.10	522.89	22	81.61	7070.76
nug15	1040.99	239918	192895.20	quit before itr 1			
nug20	estimated time: >2 months			did not run			
nug30	did not run			did not run			

(150-MHz MIPS 4400 processors with 1.5 Gbytes of RAM). For the simplex algorithm, the table lists the value of the optimal solution found (sol'n), the number of iterations taken (itr), and the total running time (time) in seconds. The default barrier uses CPLEX's crossover scheme, switching to the simplex method at the end to find a basic optimal solution. For the barrier algorithm, the table lists the value of the optimal solution found (sol'n), the number of barrier iterations taken (b-itr), the crossover time (x-time), in seconds, and the total running time (time), in seconds.

The simplex algorithm successfully found optimal solutions for the instances corresponding to quadratic assignment problems of dimension $n \leq 15$. The largest instance solved required over 53 hours of CPU time. On instance `nug20`, the simplex code took over 9 cpu days to finish phase I, and took 1,348,478 seconds (over 15 cpu days) to execute 95,960 iterations (14.05 seconds/iteration). At that point, the simplex algorithm had brought the primal objective function down to only 2916 (the optimal is 2182). Proceeding at that rate, we estimate that the simplex code would take over 63 CPU days to finish (see Figure 1). The barrier code was run with default parameters and had no problem solving all instances having QAP dimension $n \leq 8$. On `nug12`, the code using default parameters encountered numerical difficulties and did not get past the initial factorization. I. Lustig, of CPLEX Optimization, said that the LP model had many dependent rows and recommended that we use the somewhat slower, but more robust, pulling factorization option (CPLEX command `set bar cholesky 0`). The default on the SGI machines is the pushing factorization. Indeed, with the pulling factorization the barrier code was able to find the optimal solution to `nug12`. On instance `nug15`, however, both factorization options failed. I. Lustig informed us that version 3.0 of CPLEX will include a dependency option to their presolve. He expects this to help get past the numerical difficulties encountered here.

Even if the numerical difficulties encountered by the barrier method were resolved, we estimate that the barrier method would take over 15 CPU days to solve `nug20` (see Figure 1). Though this is four times faster than the simplex code, it is nevertheless impractical. The most serious problem with the barrier method arises from the very high density of the direction-determining linear systems of these instances (Table 3 lists statistics of the factors used in the Barrier method: number of rows, dimension of the dense window of the factor, number of nonzeroes in the factor). Such large densities make indirect methods preferable to direct methods for solving these linear systems.

Karmarkar and Ramakrishnan (Karmarkar & Ramakrishnan, 1991) describe an implementation of an interior point method for LP that applies the preconditioned conjugate gradient method to approximately solve the linear systems of each interior point iteration. The algorithm solves primal linear programs with upper bounds on all variables (if a variable is unbounded, a large upper bound is assigned to it) by first solving the dual program, and then in a final phase finding a feasible (optimal) primal solution that is complementary to the dual optimal found in the earlier phase. The algorithm, called Approximate Dual Projective (ADP), initially takes a series of centering steps to bring the initial iterate closer to the center of the dual LP polytope. Then, after each dual affine step,

Table 3: CPLEX 2.1 Barrier Factors on Nugent *et al.* test problems

name	rows	dense window	nz(factor)
nug05	185	128	10653
nug06	372	247	39657
nug07	602	417	105711
nug08	912	612	248166
nug12	3192	2089	2975612
nug15	6333	4273	11233305

Table 4: ADP on the Nugent *et al.* test problems

	ADP steps			ADP		CPU time ratio	
name	affine	center	CG	CPU time	sol'n	simplex/adp	barrier/adp
nug05	14	39	244	1.62	50.00	0.14	0.35
nug06	17	43	336	2.57	86.00	0.87	1.31
nug07	19	48	1059	6.24	148.00	3.52	2.83
nug08	18	45	944	9.50	203.49	9.61	6.02
nug12	29	71	16329	754.12	522.89	13.20	9.38
nug15	36	81	19095	5203.83	1040.99	37.07	-
nug20	31	59	13846	6745.46	2181.57	-	-
nug30	36	64	18547	35057.96	4804.56	-	-

the algorithm takes one or more centering steps to maintain the iterate well centered. To compute the affine and centering directions, a system of linear equations must be solved. The linear systems for the affine and centering directions differ only with respect to the right hand side vector, allowing the same preconditioner to be used for both computations. Furthermore, the preconditioner need not be computed at each iteration. Once computed, it can be reused during several iterations. The accuracy of the conjugate gradient method at each interior point iteration, as well as the cpu time of the conjugate gradient method, determine if a new, more accurate, preconditioner needs to be computed. If a primal optimal solution is required, a final series of centering steps are taken after the convergence of the dual iterates, prior to the primal feasibility phase.

Since ADP solves the dual of the linear program, any feasible dual iterate produced by ADP is a lower bound for the QAP. For the purpose of finding a lower bound for QAP, we do not need to compute the primal solution, and thus terminate the algorithm with only a dual optimal solution. Table 4 summarizes the results of running ADP on the Nugent *et al.* class of test problems. For purpose of comparison, the table also lists CPU time ratios with respect to the two CPLEX codes. Figure 1 plots running times for CPLEX primal simplex, CPLEX barrier, and ADP for the Nugent *et al.* test problems. The estimated running time for the simplex code on `nug20` is also plotted, since the algorithm does make progress towards the optimal. We do not plot our estimate for the barrier algorithm, since even for `nug15` it is not able to get past the first factorization.

Figure 2 shows the primal and dual objective function values generated, respectively, by the CPLEX primal simplex algorithm and the ADP algorithm on instance `nug15`. The plot shows only feasible iterates. Note that the simplex code solves a perturbed problem, gaining in objective function value near the end, while unperturbing the data. Since all dual feasible ADP iterates are lower bounds, it is interesting to observe that though the algorithm took over 5000 seconds to stop, it was within 10% of the optimal after only 77 seconds, within 5% in 88 seconds, and within 1%

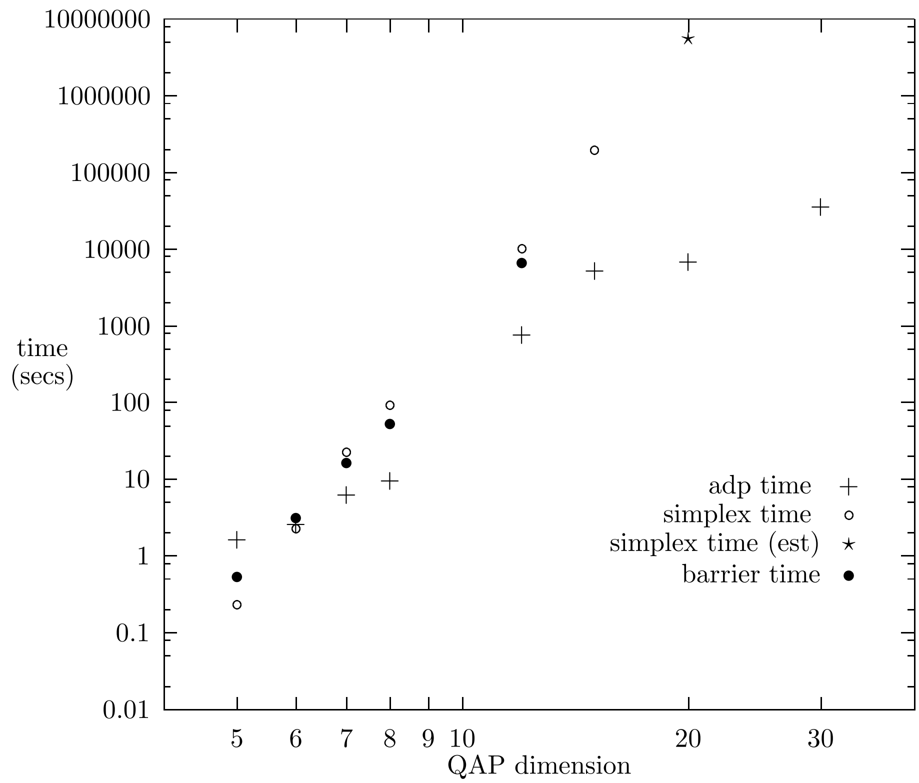

Figure 1: CPLEX and ADP solvers on Nugent *et al.* problems

in 168 seconds. In 520.35 seconds it attained an objective function value of 1040.009084, which rounded up gives the QAPLP lower bound of 1041.

3 Computational Results

In this section we report on computational results. We use as our testbed the suite of QAP test problems QAPLIB. Because of memory requirements, we limit our experiment to the 63 instances in QAPLIB that are of dimension n less than or equal to 30. For a QAP of dimension $n = 30$, ADP requires about 800 Mbytes of memory (using the ADP parameter settings described later in this section).

The experiment was done on a Silicon Graphics Challenge, whose hardware description is displayed in Figure 3 by executing the `hinv` command. The ADP code is written in C and Fortran. It was compiled with the `cc` and `f77` compilers using compiler flags `CFLAGS = -O -DVAX -cckr -p` and `FFLAGS = -O2 -p -trapuv`. Running times were measured by making the system call `times` and converting to seconds, using the `HZ` defined in `sys/param.h`.

The ADP code requires a number of parameters to be set. First, we describe some global parameters. The fraction of the maximum step length of the dual and centering steps are set to `fract` $= 0.8$ and `pfract` $= 0.5$, respectively. The dual iterates are said to be converged when the relative dual objective function improvement falls below `stptol` $= 10^{-6}$. The number of nonzero elements in the factors of the preconditioner is limited to be at most `nfacl` times the number of nonzero elements in the linear program constraint matrix A. We have set `nfacl` $= 10$. If this limit is reached, the algorithm terminates.

Since the computation of the centering direction is not exact, the centering step may be limited by a dual constraint, not permitting a step size fraction of `pfract` to be taken. Centering steps are repeated until the sum of lengths taken over all centering steps (of the current interior point iteration) exceeds `thrsp` $= 0.95$.

Next, we describe some conjugate gradient parameters. The conjugate gradient iterations terminate when the solution residue falls below a certain value and the angle between the computed direction and the right hand side vector is small. Suppose the system begin solved is $Bx = b$ and the current CG solution is $\hat{x}$. At every CG iteration, the residue $\|b - B\hat{x}\|/\|b\|$ is computed. At every CG iteration after the first CG iteration in which the residue falls below `tolres` $= 10^{-2}$ for the affine direction computation (`ptolres` $= 10^{-1}$ for the centering computation), the cosine of the angle θ between $\hat{x}$ and b is computed. The conjugate gradient terminates when the angle θ is such that $|1 - \cos\theta| <$ `coslim` for the affine step computation and $|1 - \cos\theta| <$ `pcoslim` for the centering step computation. The parameters `coslim` and `pcoslim` are initially (in the first interior point iteration) set to 10^{-3} and 10^{-1}, respectively. At each interior point iteration, they are tightened, according to `coslim` $=$ `gcoslim` $\times$ `coslim` and `pcoslim` $=$ `gpcoslim` $\times$ `gcoslim`. The values of `gcoslim` and `gpcoslim` are both set to 0.95. If `nitr` $= 800$ CG iterations are taken without satisfying the stopping criterion, a new preconditioner is computed, and the CG restarts from where it left off.

Finally, we give some preconditioner parameters. The basis for the use of preconditioners in ADP is that if many nonzero elements of the matrices A and later $AD^2A^\top$ are dropped, little fill-in will result in the computation of the preconditioner, i.e. the factors of the approximate $AD^2A^\top$. On the other hand, if too many elements are dropped, the preconditioner will not be effective, resulting in a large number of conjugate gradient iterations. As each new preconditioner is computed, fewer nonzero elements are dropped, making each new preconditioner more effective, but also more expensive to compute and to apply in the conjugate gradient algorithm. Four parameters control dropping of small nonzero elements. The parameter `dpre` ($0 \leq$ `dpre` ≤ 1) controls dropping of nonzero elements of the constraint matrix A. A value of 1 forces all nonzero elements of A to be dropped, while a value of 0 drops no element. `dpre` is initially set to 0.8. As each new preconditioner is computed, `dpre` is decreased in value, according to `dpre` $=$ `dpre` $\times$ `gdpre`. In these experiments, the parameter `gdpre` is set to 0.5. Similarly to `dpre` and `gdpre`, parameters `dpost` and `gpost`

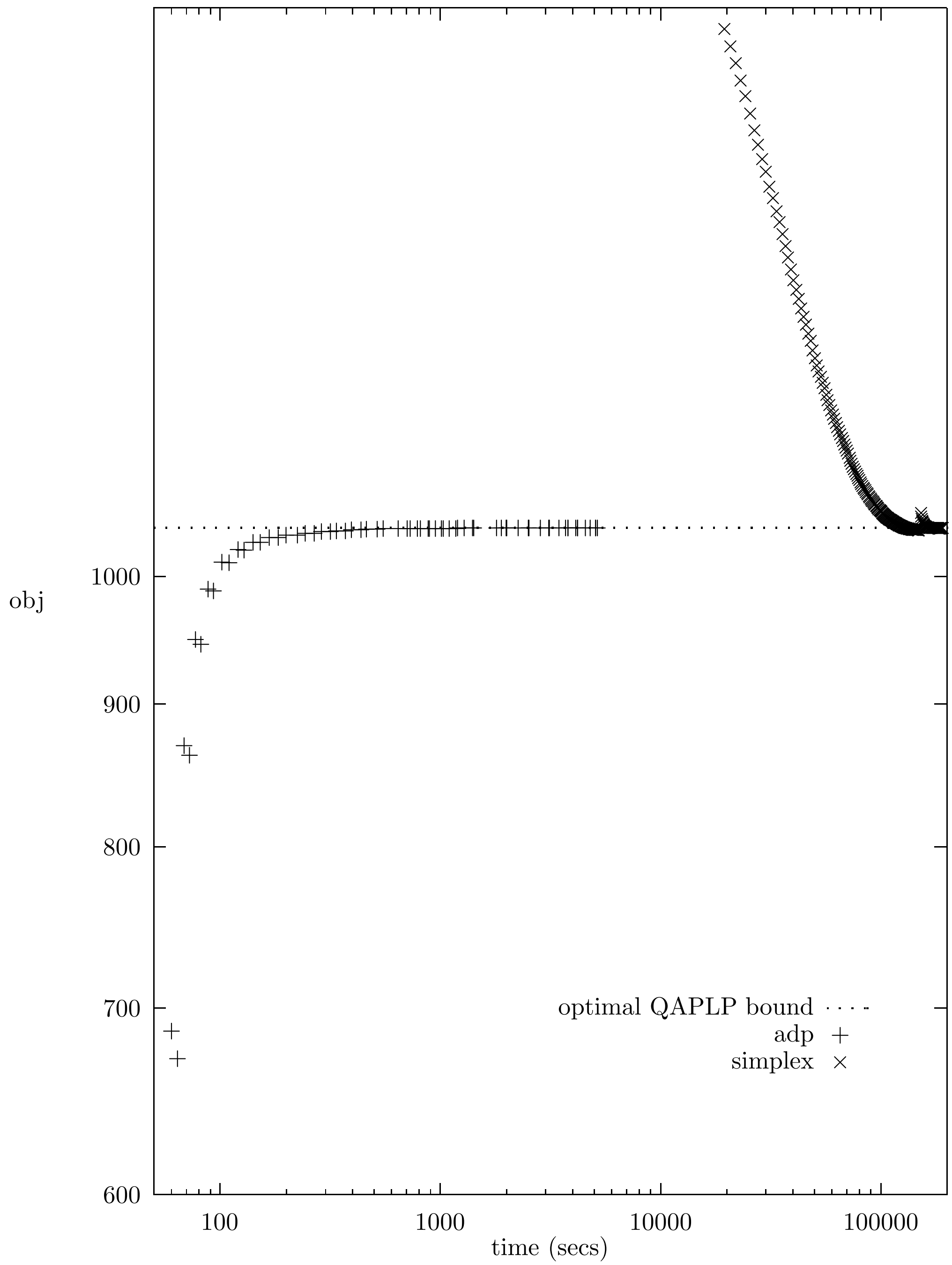

Figure 2: CPLEX simplex and ADP iterates on `nug15`

```
16 150 MHZ IP19 Processors
CPU: MIPS R4400 Processor Chip Revision: 5.0
FPU: MIPS R4010 Floating Point Chip Revision: 0.0
Data cache size: 16 Kbytes
Instruction cache size: 16 Kbytes
Secondary unified instruction/data cache size: 1 Mbyte
Main memory size: 1536 Mbytes, 8-way interleaved
I/O board, Ebus slot 13: IO4 revision 1
I/O board, Ebus slot 15: IO4 revision 1
Integral EPC serial ports: 8
Integral Ethernet controller: et1, Ebus slot 13
Integral Ethernet controller: et0, Ebus slot 15
FDDIXPress controller: ipg0, version 1
Integral SCSI controller 131: Version WD33C95A
Disk drive: unit 4 on SCSI controller 131
Disk drive: unit 3 on SCSI controller 131
Disk drive: unit 2 on SCSI controller 131
Disk drive: unit 1 on SCSI controller 131
Integral SCSI controller 130: Version WD33C95A
Tape drive: unit 7 on SCSI controller 130: 8mm(8500) cartridge
Jukebox: unit 6 on SCSI controller 130
Disk drive: unit 4 on SCSI controller 130
Disk drive: unit 3 on SCSI controller 130
Disk drive: unit 2 on SCSI controller 130
Disk drive: unit 1 on SCSI controller 130
Integral SCSI controller 4: Version WD33C95A
Disk drive: unit 4 on SCSI controller 4
Disk drive: unit 3 on SCSI controller 4
Disk drive: unit 2 on SCSI controller 4
Disk drive: unit 1 on SCSI controller 4
Integral SCSI controller 3: Version WD33C95A
Disk drive: unit 4 on SCSI controller 3
Disk drive: unit 3 on SCSI controller 3
Disk drive: unit 2 on SCSI controller 3
Integral SCSI controller 2: Version WD33C95A
Tape drive: unit 7 on SCSI controller 2: 8mm(8500) cartridge
Jukebox: unit 6 on SCSI controller 2
Disk drive: unit 5 on SCSI controller 2
Disk drive: unit 4 on SCSI controller 2
Disk drive: unit 3 on SCSI controller 2
Disk drive: unit 2 on SCSI controller 2
Integral SCSI controller 1: Version WD33C95A
Disk drive: unit 1 on SCSI controller 1
Integral SCSI controller 0: Version WD33C95A
Tape drive: unit 7 on SCSI controller 0: QIC 150
CDROM: unit 6 on SCSI controller 0
Disk drive: unit 2 on SCSI controller 0
Disk drive: unit 1 on SCSI controller 0
Integral EPC parallel port: Ebus slot 13
Integral EPC parallel port: Ebus slot 15
VME bus: adapter 0 mapped to adapter 61
VME bus: adapter 61
```

Figure 3: Computer hardware configuration

Table 5: QAPLP statistics: QAP dim $5 \leq n \leq 10$

name	n	bks	qaplp	$\frac{\text{bks}-\text{qaplp}}{\text{bks}}$	glb	bklb	is qaplp best ?
nug05	5	50	50	0.0000	50	50	y(m)
nug06	6	86	86	0.0000	84	86	y(m)
nug07	7	148	148	0.0000	137	148	y(m)
esc08a	8	2	0	1.0000	0	0	y(m)
esc08b	8	8	2	0.7500	1	1	y(i)
esc08c	8	32	22	0.3125	13	13	y(i)
esc08d	8	6	2	0.6667	2	2	y(m)
esc08e	8	2	0	1.0000	0	0	y(m)
esc08f	8	18	18	0.0000	9	9	y(i)
nug08	8	214	204	0.0467	186	194	y(i)
lipa10a	10	473	473	0.0000	463	463	y(i)
lipa10b	10	2008	2008	0.0000	2008	2008	y(m)
rou10	10	174220	170384	0.0220	152886	152886	y(i)
scr10	10	26992	26874	0.0044	24297	24297	y(i)

dynamically control the fill-in that occurs in the preconditioner, by dropping small nonzero elements of the $AD^2A^\top$ matrix. Parameter `dpost` is initially set to 0.8, while parameter `gdpost` is set to 0.5. The pivot tolerance parameter $\texttt{lndp} = 10^{-12}$ is used to decide linear dependency of rows in the preconditioner.

The row ordering heuristic used is minimum degree ordering. It is not computed every time a factorization is done. Reordering is only done if the number of nonzero elements in the new preconditioner is more than $1+\texttt{mndg}$ times the number of nonzero elements in the previous preconditioner. We use $\texttt{mndg} = 0.5$. The maximum size of the dense window data structure for the factors is set to $\texttt{dnswndw} = 4000$.

Tables 5–8 summarize the quality of the QAPLP bounds produced by ADP. For each instance, the tables list its QAPLIB name, QAP dimension (n), cost value of best known solution (bks), the QAPLP value (qaplp), the relative error of the QAPLP with respect to the best known solution, the Gilmore-Lawler lower bound (glb), the best known lower bound prior to the QAPLP bound (bklb), and an indication if the QAPLP bound is the best known for this instance (n = no; y(i) = yes, QAPLP improved best known lower bound; y(m) = yes, QAPLP matched best known lower bound).

Tables 9–12 show solution statistics for the ADP code. For each instances, the tables give the instance name, the number of ADP affine scaling steps (aff), the number of centering steps (pot), the number of refactorizations (rfac), the total number of conjugate gradient steps (cg), the total cpu time (in seconds) associated with reordering prior to refactorization, symbolic factorization, numeric factorization, and conjugate gradient method, as well as the overall solution time, how ADP terminated (stop: do = relative improvement of dual objective function; nz = number of nonzero elements in preconditioner), and fractional dual objective function value.

Figure 4 plots ADP running times (in seconds) as a function of QAP dimension, to compute the QAPLP lower bound, as well as to compute lower bounds within 1%, 5%, and 10% of the QAPLP lower bound.

Figures 5–8 illustrate how the dual objective function approaches the best QAPLP lower bound, as a function of cpu time. These plots suggest that it is often the case that many interior point iterations are spent trying to improve on the last digits of accuracy, when the lower bound already produced is sufficiently good. In a branch and bound method, one need not run ADP with such a tight interior point stopping criterion.

Tables 13–16 list running times for ADP to reach an objective function value, that rounded up provides the Gilmore-Lawler lower bound and the QAPLP bound, as well as the ratios to the total ADP running time to reach those bounds.

We make the following observations regarding the computational results.

Table 6: QAPLP statistics: QAP dim $12 \le n \le 16$

name	n	bks	qaplp	$\frac{\text{bks}-\text{qaplp}}{\text{bks}}$	glb	bklb	is qaplp best ?
chr12a	12	9552	9552	0.0000	7245	7245	y(i)
chr12b	12	9742	9742	0.0000	7146	7146	y(i)
chr12c	12	11156	11156	0.0000	7976	7976	y(i)
nug12	12	578	523	0.0952	493	528	n
rou12	12	235528	224278	0.0478	202272	202272	y(i)
scr12	12	31410	29827	0.0504	27858	27858	y(i)
chr15a	15	9896	9511	0.0389	5625	5625	y(i)
chr15b	15	7990	7990	0.0000	4653	4653	y(i)
chr15c	15	9504	9504	0.0000	6165	6165	y(i)
nug15	15	1150	1041	0.0948	963	1083	n
rou15	15	354210	324869	0.0828	298548	298548	y(i)
scr15	15	51140	49264	0.0367	44737	44737	y(i)
esc16a	16	68	48	0.2941	38	47	y(i)
esc16b	16	292	278	0.0479	220	250	y(i)
esc16c	16	160	118	0.2625	83	95	y(i)
esc16d	16	16	4	0.7500	3	3	y(i)
esc16e	16	28	14	0.5000	12	12	y(i)
esc16g	16	26	14	0.4615	12	12	y(i)
esc16h	16	996	704	0.2932	625	708	n
esc16i	16	14	0	1.0000	0	0	y(m)
esc16j	16	8	2	0.7500	1	1	y(i)

Table 7: QAPLP statistics: QAP dim $18 \le n \le 22$

name	n	bks	qaplp	$\frac{\text{bks}-\text{qaplp}}{\text{bks}}$	glb	bklb	is qaplp best ?
chr18a	18	11098	10752	0.0312	6779	6779	y(i)
chr18b	18	1534	1534	0.0000	1534	1534	y(m)
els19	19	17212548	16874205	0.0197	11971949	11971949	y(i)
chr20a	20	2192	2174	0.0082	2150	2150	y(i)
chr20b	20	2298	2287	0.0048	2196	2196	y(i)
chr20c	20	14142	14142	0.0000	8601	8601	y(i)
lipa20a	20	3683	3683	0.0000	3667	3667	y(i)
lipa20b	20	27076	27076	0.0000	27076	27076	y(m)
nug20	20	2570	2182	0.1510	2057	2394	n
rou20	20	725522	643346	0.1133	599948	599948	y(i)
scr20	20	110030	95113	0.1356	86766	87968	y(i)
chr22a	22	6156	6143	0.0021	5924	5924	y(i)
chr22b	22	6194	6181	0.0021	5936	5936	y(i)

Table 8: QAPLP statistics: QAP dim $25 \le n \le 30$

name	n	bks	qaplp	$\frac{\text{bks}-\text{qaplp}}{\text{bks}}$	glb	bklb	is qaplp best ?
chr25a	25	3796	3785	0.0029	2765	2765	y(i)
bur26a	26	5426670	5334208	0.0170	5310923	5310923	y(i)
bur26b	26	3817852	3736954	0.0212	3710681	3710681	y(i)
bur26c	26	5426795	5359110	0.0125	5318021	5318021	y(i)
bur26d	26	3821225	3705831	0.0302	3717706	3717706	n
bur26e	26	5386879	5315311	0.0133	5307361	5307361	y(i)
bur26f	26	3782044	3712627	0.0184	3707226	3707226	y(i)
bur26g	26	10663354	10047627	0.0577	9979718	9979718	y(i)
bur26h	26	7560690	7036448	0.0693	6975151	6975151	y(i)
kra30a	30	88900	76003	0.1451	68360	68360	y(i)
kra30b	30	91420	76752	0.1604	69065	69065	y(i)
lipa30a	30	13178	12401	0.0590	13147	13147	n
lipa30b	30	151426	151426	0.0000	151426	151426	y(m)
nug30	30	6124	4805	0.2154	4539	5772	n
tho30	30	149936	100784	0.3278	90578	136447	n

Table 9: ADP statistics: QAP dim $5 \le n \le 12$

	adp steps				adp time (secs)					adp sol'n	
name	aff	pot	rfac	cg	ord	sfac	nfac	cg	tot	stop	obj
nug05	14	30	15	244	0.0	0.0	0.0	0.3	1.6	do	50.0
nug06	17	34	36	336	0.2	0.0	0.0	0.9	2.6	do	86.0
nug07	19	39	45	1059	0.1	0.3	0.1	3.6	6.2	do	148.0
esc08a	18	41	42	689	0.3	0.3	0.2	4.4	8.6	do	0.0
esc08b	18	38	40	873	0.3	0.0	0.2	5.2	9.3	do	2.0
esc08c	18	37	44	698	0.0	0.0	0.2	4.3	8.4	do	22.0
esc08d	18	37	42	545	0.3	0.3	0.2	4.0	8.1	do	2.0
esc08e	19	40	43	969	0.3	0.1	0.2	5.6	9.8	do	0.0
esc08f	17	36	38	823	0.2	0.2	0.1	5.2	9.2	do	18.0
nug08	18	36	41	944	0.3	0.0	0.2	5.7	9.5	do	203.5
lipa10a	18	38	37	531	0.8	0.0	0.3	9.1	17.2	do	946.0
lipa10b	17	37	37	432	0.5	0.2	0.4	7.7	15.6	do	4016.0
rou10	20	42	51	5178	0.9	0.4	0.5	67.0	86.6	do	170383.6
scr10	36	69	87	19404	4.5	0.7	1.1	350.0	466.4	do	26873.0

Table 10: ADP statistics: QAP dim $12 \le n \le 16$

	adp steps				adp time (secs)					adp sol'n	
name	aff	pot	rfac	cg	ord	sfac	nfac	cg	tot	stop	obj
chr12a	25	59	68	2985	2.6	1.0	1.4	100.8	152.1	do	9551.9
chr12b	23	57	58	2208	1.8	0.8	1.2	71.7	120.6	do	9741.9
chr12c	25	59	64	3732	2.0	0.9	1.4	120.8	171.7	do	11155.8
nug12	28	62	73	16206	10.4	1.2	2.2	598.2	754.1	nz	522.9
rou12	22	43	55	5793	2.3	0.8	1.1	164.8	207.8	do	224277.4
scr12	30	59	74	10234	4.5	1.1	1.9	327.0	448.3	nz	29826.5
chr15a	39	75	93	17887	15.8	3.8	6.5	1623.2	2042.4	do	9510.4
chr15b	31	65	79	8871	8.5	3.0	4.5	704.5	910.1	do	7989.8
chr15c	27	63	69	4475	6.4	2.5	3.7	357.4	491.3	do	9503.8
nug15	35	72	82	18967	135.1	3.5	6.2	1862.6	5203.8	nz	1041.0
rou15	22	46	58	6332	5.9	2.1	3.3	479.1	595.5	do	324868.9
scr15	35	62	75	13219	48.2	3.1	4.8	1181.8	2244.9	nz	49263.7
esc16a	20	43	50	2319	5.5	1.9	3.2	244.1	315.0	do	48.0
esc16b	17	39	44	1015	4.4	1.6	2.7	115.8	178.3	do	278.0
esc16c	20	40	49	1308	0.8	0.3	10.1	450.8	846.1	do	118.0
esc16d	21	46	54	2761	6.2	2.5	3.7	286.1	362.3	do	4.0
esc16e	21	44	51	2234	5.7	1.9	3.3	234.2	306.9	do	14.0
esc16g	21	44	51	2085	5.4	1.9	3.4	212.9	283.5	do	14.0
esc16h	21	46	55	2058	5.7	2.6	4.0	216.4	290.7	do	704.0
esc16i	23	49	58	2161	5.9	2.5	4.0	221.0	300.7	do	0.0
esc16j	23	49	58	2899	5.9	2.3	4.0	291.7	371.2	do	2.0

Table 11: ADP statistics: QAP dim $18 \le n \le 22$

	adp steps				adp time (secs)					adp sol'n	
name	aff	pot	rfac	cg	ord	sfac	nfac	cg	tot	stop	obj
chr18a	40	78	97	19680	31.2	8.1	13.1	3595.7	4333.5	do	10751.1
chr18b	27	55	69	7953	12.8	4.7	8.1	1299.7	1725.6	nz	1533.9
els19	49	90	115	21329	117.9	12.6	20.5	5245.3	8270.2	do	16874204.8
chr20a	47	86	94	21196	136.1	12.0	20.5	6627.1	9281.0	do	2173.8
chr20b	35	73	85	13877	26.0	10.1	17.0	3649.1	4603.6	do	2286.9
chr20c	51	76	97	19460	74.1	12.2	20.8	5388.1	6798.1	do	14142.0
lipa20a	22	45	53	2385	14.2	5.4	8.8	621.6	806.8	do	7366.0
lipa20b	20	43	46	920	12.0	4.2	7.2	264.2	432.2	do	54152.0
nug20	27	53	65	10952	31.4	6.8	11.3	2797.8	3611.4	nz	2181.4
rou20	27	52	65	13326	37.8	7.0	12.1	3453.2	4427.6	do	643345.5
scr20	39	68	79	13905	203.9	10.4	18.7	4192.9	8303.8	nz	95112.3
chr22a	70	103	149	41657	227.3	30.1	50.2	19665.8	24118.5	do	6142.7
chr22b	72	104	151	32444	195.7	30.0	51.9	16257.7	20224.8	do	6180.7

Table 12: ADP statistics: QAP dim $25 \leq n \leq 30$

	adp steps				adp time (secs)					adp sol'n	
name	aff	pot	rfac	cg	ord	sfac	nfac	cg	tot	stop	obj
chr25a	76	109	137	44958	1162.3	45.9	80.0	38640.3	64587.8	do	3784.3
bur26a	23	54	49	4603	45.0	16.8	29.6	4001.1	5410.6	nz	5334207.9
bur26b	25	59	56	8422	53.9	20.5	35.7	6968.8	8387.1	nz	3736953.1
bur26c	35	64	72	11175	128.4	29.3	49.8	9057.0	11728.1	nz	5359109.4
bur26d	18	47	39	4278	34.2	12.8	21.9	3710.7	4901.5	nz	3705830.0
bur26e	24	59	54	5743	51.5	19.1	34.1	4733.2	6176.1	nz	5315310.5
bur26f	22	50	46	5272	42.5	16.1	26.8	4497.9	5874.8	nz	3712626.5
bur26g	29	59	58	6373	55.8	20.5	36.7	5506.8	6957.4	nz	10047626.5
bur26h	28	56	56	6241	53.0	19.9	35.7	5425.6	6896.2	nz	7036447.8
kra30a	32	60	78	15761	223.3	45.5	76.0	20834.0	24679.0	nz	76002.3
kra30b	37	63	87	19139	502.8	53.7	90.7	26160.1	30481.8	nz	76751.8
lipa30a	23	53	61	5182	37.7	6.9	12.3	1493.9	2446.2	nz	24801.9
lipa30b	20	47	50	1574	64.0	23.8	39.5	2139.2	3150.4	do	302851.9
nug30	34	60	72	16802	356.2	42.2	71.8	22158.3	28819.6	nz	4804.4
tho30	37	63	87	18020	616.7	53.0	90.0	23803.1	28697.4	nz	100784.0

Table 13: ADP statistics: time to reach bounds: QAP dim $5 \leq n \leq 10$

	to reach GLB		to reach QAPLP	
name	time (secs)	ratio to tot time	time (secs)	ratio to tot time
nug05	1.5	0.914	1.5	0.914
nug06	2.0	0.790	2.2	0.852
nug07	3.2	0.508	4.1	0.651
esc08a	5.1	0.598	5.1	0.598
esc08b	4.5	0.488	5.0	0.534
esc08c	4.7	0.558	5.5	0.652
esc08d	5.2	0.651	5.2	0.651
esc08e	5.2	0.523	5.2	0.523
esc08f	4.2	0.459	5.7	0.624
nug08	4.6	0.481	7.6	0.804
lipa10a	7.0	0.407	7.0	0.407
lipa10b	7.0	0.448	7.0	0.448
rou10	10.2	0.117	85.6	0.988
scr10	15.1	0.032	460.7	0.988

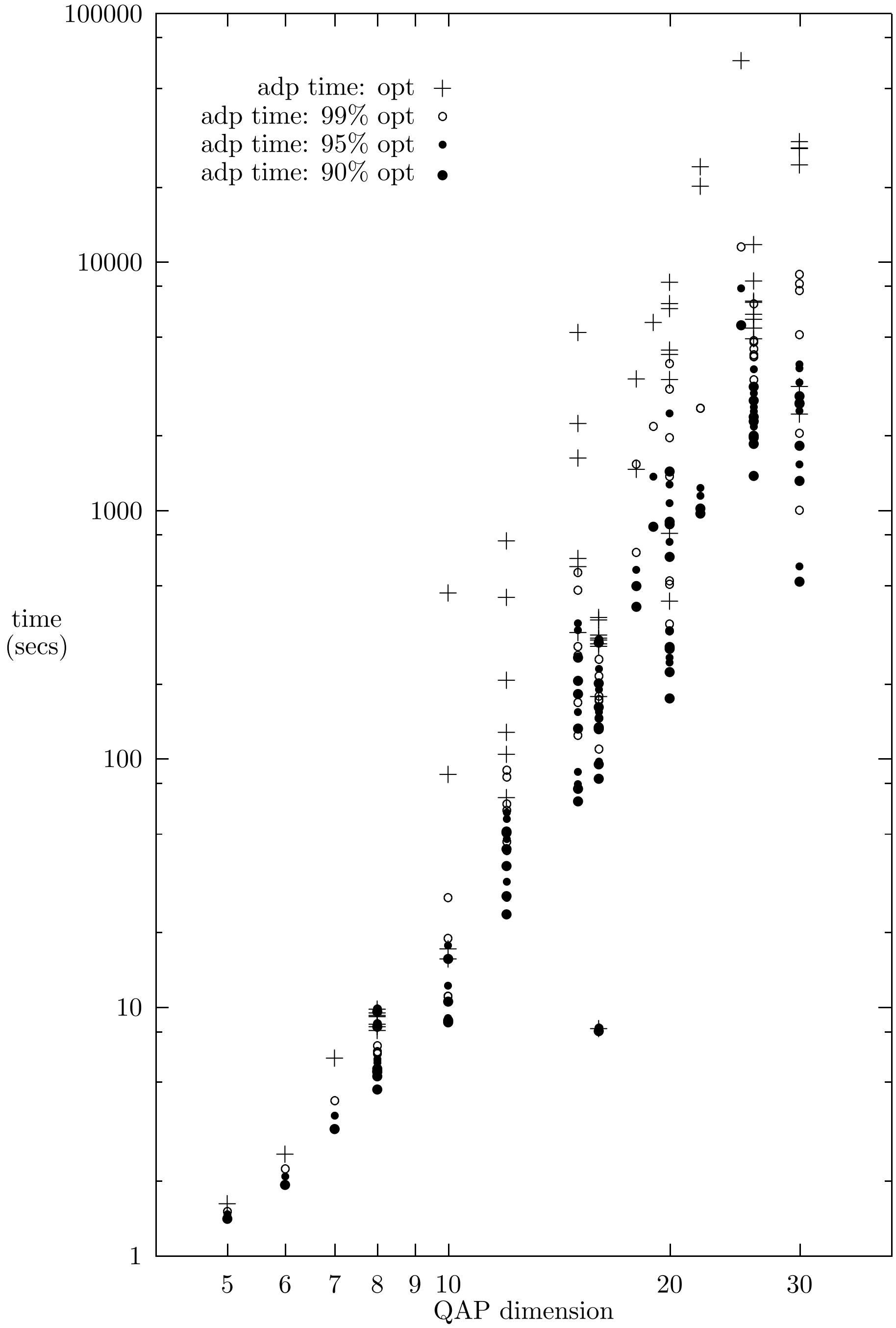

Figure 4: ADP times for approximate and exact solutions

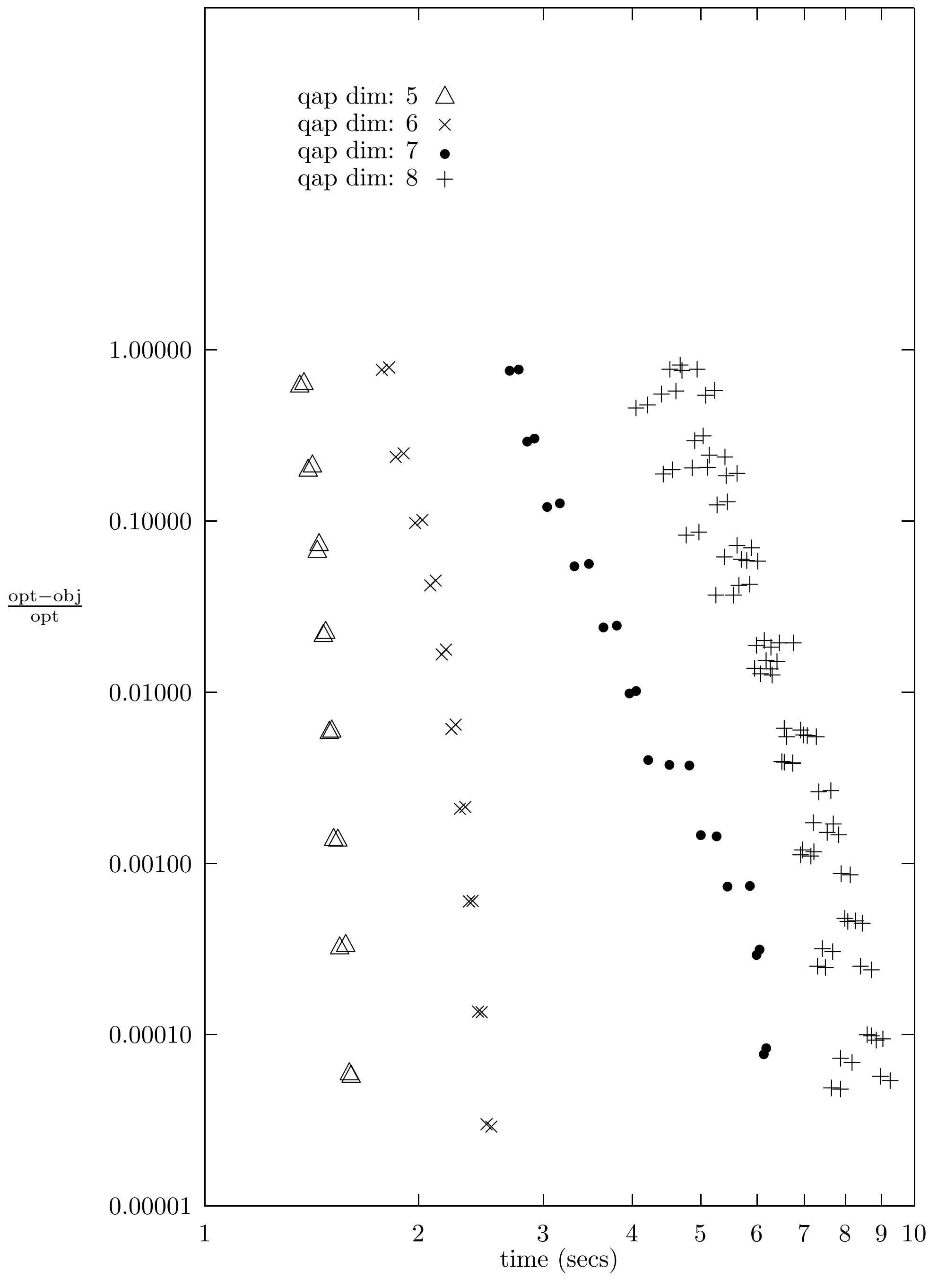

Figure 5: QAPLP bounds as a function of CPU time: (dim: $5 \leq n \leq 8$)

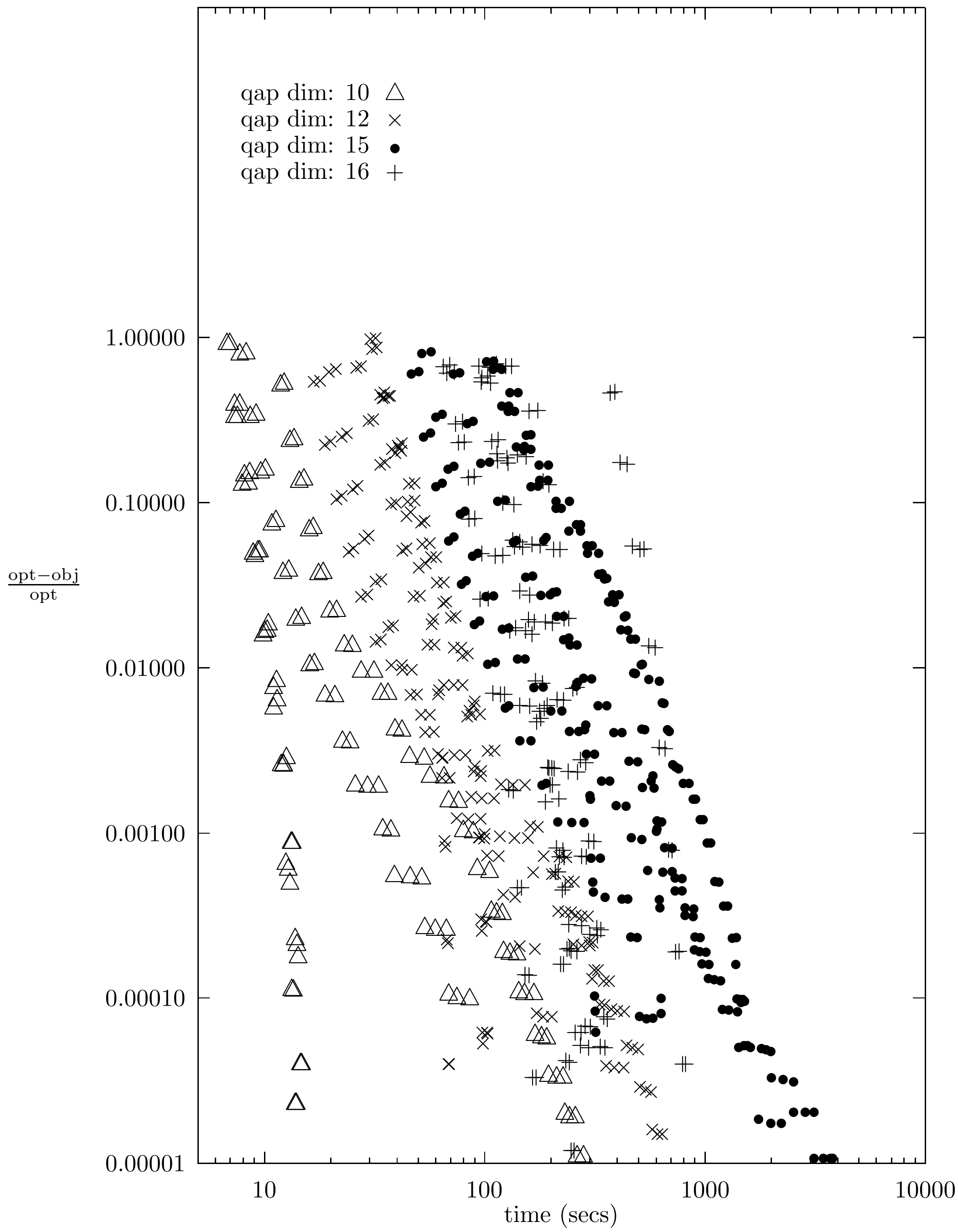

Figure 6: QAPLP bounds as a function of CPU time: (dim: $10 \leq n \leq 16$)

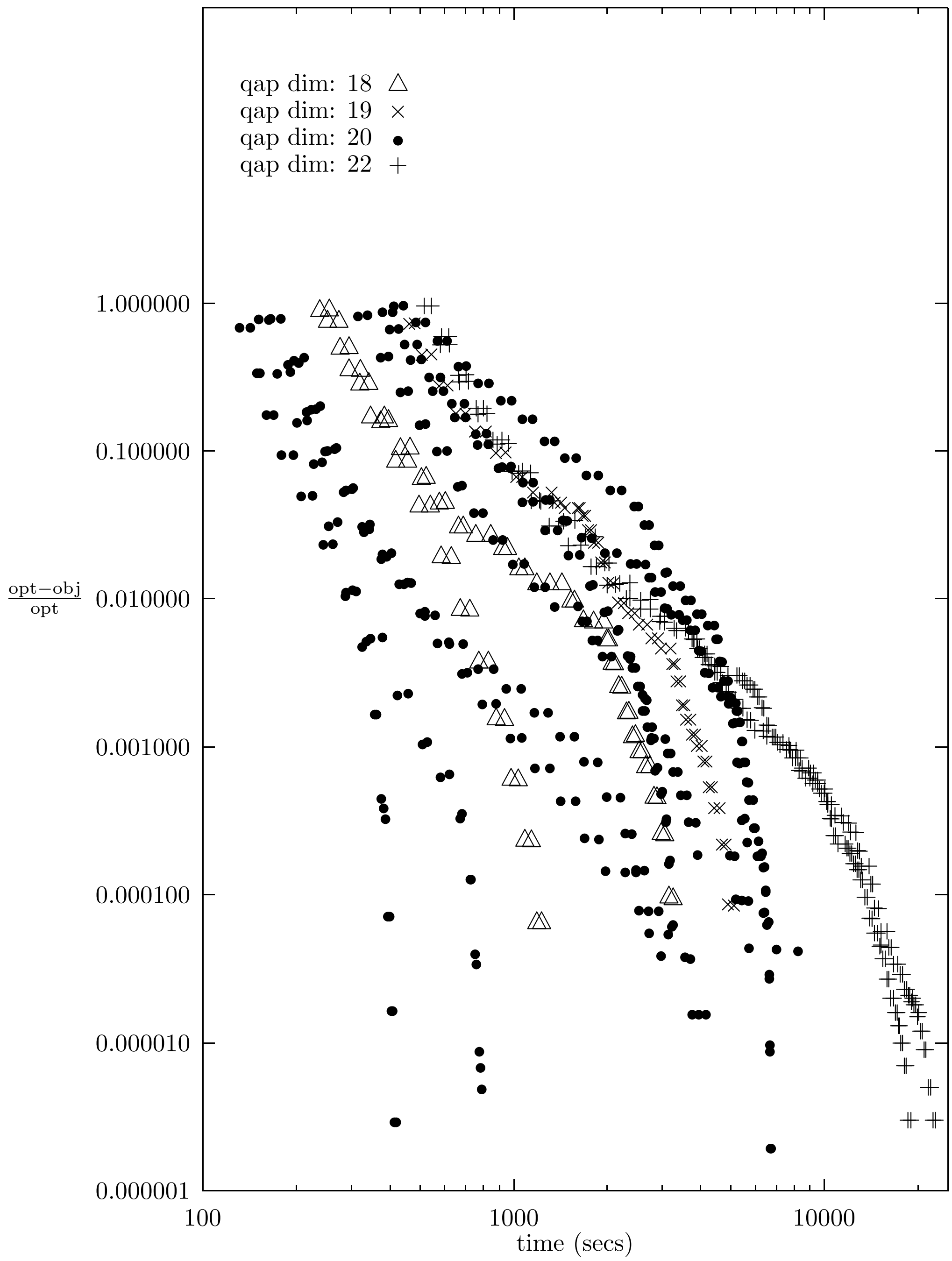

Figure 7: QAPLP bounds as a function of CPU time: (dim: $18 \leq n \leq 22$)

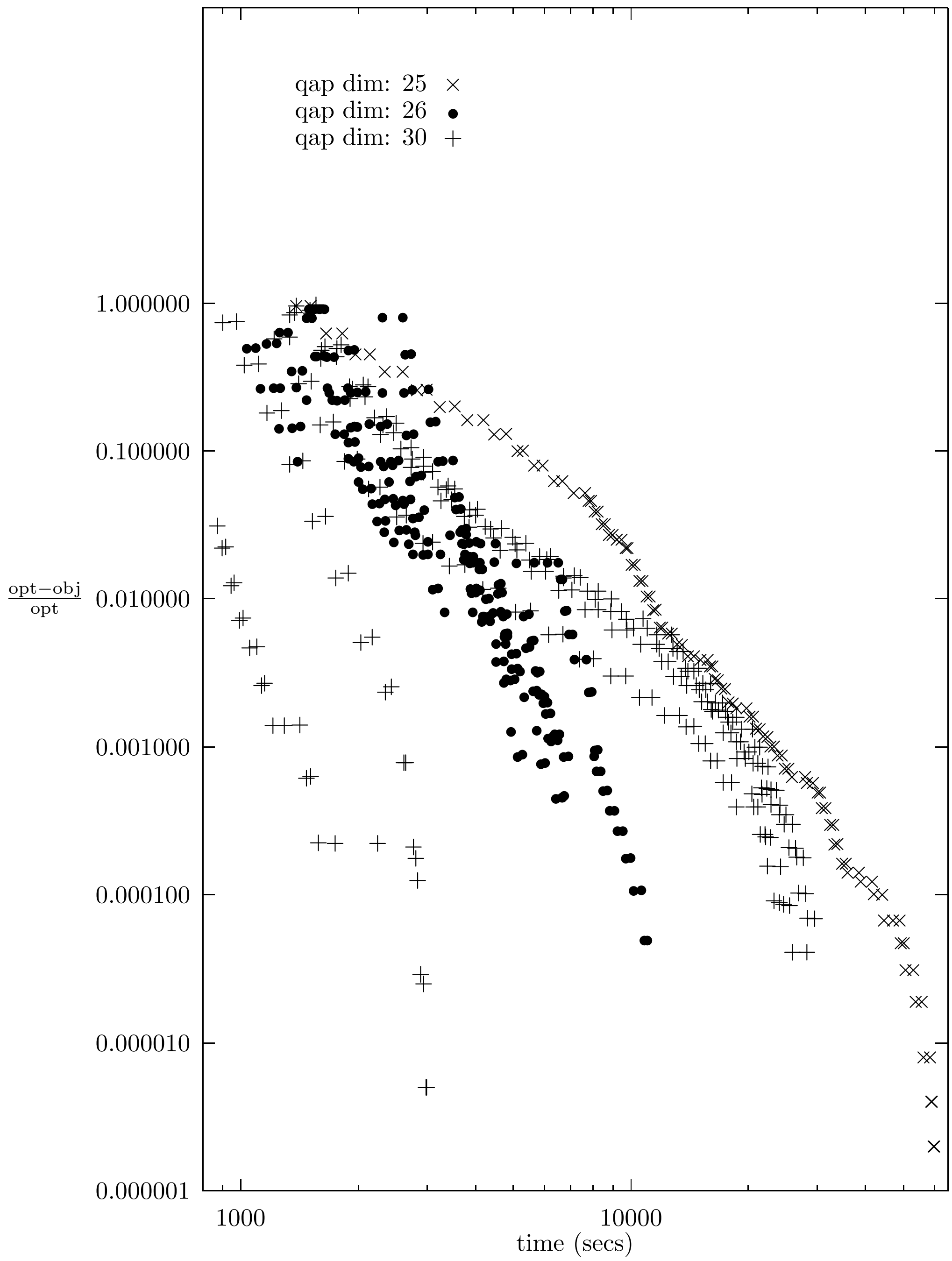

Figure 8: QAPLP bounds as a function of CPU time: (dim: $25 \leq n \leq 30$)

Table 14: ADP statistics: time to reach bounds: QAP dim $12 \le n \le 16$

	to reach GLB		to reach QAPLP	
name	time (secs)	ratio to tot time	time (secs)	ratio to tot time
chr12a	37.5	0.246	97.5	0.641
chr12b	35.2	0.292	68.3	0.566
chr12c	36.9	0.215	102.5	0.597
nug12	29.9	0.040	82.3	0.109
rou12	22.6	0.109	202.7	0.975
scr12	39.8	0.089	429.1	0.957
chr15a	142.3	0.070	1521.2	0.745
chr15b	117.7	0.129	625.2	0.687
chr15c	129.2	0.263	316.5	0.644
nug15	81.8	0.016	464.9	0.089
rou15	64.3	0.108	582.4	0.978
scr15	124.3	0.055	2231.7	0.994
esc16a	115.4	0.366	160.8	0.510
esc16b	81.2	0.456	123.8	0.694
esc16c	7.9	0.009	7.9	0.009
esc16d	133.1	0.367	133.1	0.367
esc16e	106.4	0.347	126.5	0.412
esc16g	107.2	0.378	128.5	0.453
esc16h	90.2	0.310	204.4	0.703
esc16i	122.4	0.407	122.4	0.407
esc16j	150.7	0.406	150.7	0.406

Table 15: ADP statistics: time to reach bounds: QAP dim $18 \le n \le 22$

	to reach GLB		to reach QAPLP	
name	time (secs)	ratio to tot time	time (secs)	ratio to tot time
chr18a	295.8	0.068	3268.0	0.754
chr18b	1040.0	0.603	1040.0	0.603
els19	546.0	0.066	5228.0	0.632
chr20a	2993.9	0.323	5909.6	0.637
chr20b	1157.8	0.251	3018.7	0.656
chr20c	611.5	0.090	6503.0	0.957
lipa20a	142.7	0.177	142.7	0.177
lipa20b	166.0	0.384	166.0	0.384
nug20	270.2	0.075	2409.9	0.667
rou20	266.0	0.060	4173.7	0.943
scr20	613.2	0.074	8260.7	0.995
chr22a	1344.9	0.056	14385.1	0.596
chr22b	1215.7	0.060	13336.2	0.659

Table 16: ADP statistics: time to reach bounds: QAP dim $25 \leq n \leq 30$

	to reach GLB		to reach QAPLP	
name	time (secs)	ratio to tot time	time (secs)	ratio to tot time
chr25a	2610.7	0.040	44227.3	0.685
bur26a	4839.7	0.894	5284.6	0.977
bur26b	6859.9	0.818	8260.5	0.985
bur26c	4656.2	0.397	11288.8	0.963
bur26d	did not achieve GLB		4812.7	0.982
bur26e	5761.8	0.933	6050.0	0.980
bur26f	4746.1	0.808	5749.2	0.979
bur26g	5487.6	0.789	6762.3	0.972
bur26h	4015.8	0.582	6705.7	0.972
kra30a	2509.0	0.102	24144.7	0.978
kra30b	2474.1	0.081	29666.0	0.973
lipa30a	did not achieve GLB		382.2	0.156
lipa30b	979.6	0.311	979.6	0.311
nug30	2284.3	0.079	25589.5	0.888
tho30	2745.7	0.096	28162.1	0.981

- We computed QAPLP lower bounds for all instances of QAPLIB having dimension $n \leq 30$. There are 63 such instances, the largest of which have corresponding linear programs with 52,260 constraints and 379,350 variables.
- Since QAPLP is known to be at least as good as the Gilmore-Lawler lower bound, solving the LP relaxations to optimality must yield a lower bound at least as good as the Gilmore-Lawler bound. The ADP solver produced bounds at least as good as the Gilmore-Lawler lower bound for all but two instances (`bur26d` and `lipa30a`). In those two cases, the algorithm terminated with the nonzero elements in preconditioner stopping criterion, prior to the time that the relative dual objective function improvement criterion was satisfied. The cutoff used in these experiments for number of nonzero elements in the preconditioner was set to 10 times the number of nonzero elements in the A matrix. In 53 of the 63 instances the QAPLP was better than the Gilmore-Lawler lower bound.
- In 28 instances, the QAPLP bound was at least 10% greater than the GLB. In 18 it was at least 25% greater; in 9 at least 50% greater; and in three instances the QAPLP bound was twice the GLB.
- Compared to the best bounds reported in the literature, the QAPLP bound was best for all but 9 of the 63 instances considered. The 9 instances are summarized in Table 17. However, the best known lower bound for one of the 9 instances (`nug08`) was achieved with the linear programming technique described in this paper on data preprocessed with the technique described in (Chakrapani & Skorin-Kapov, 1994).
- The QAPLP bounds improved the best known lower bound for 44 of the 63 instances considered.
- In 15 instances the QAPLP bound was tight, i.e. equaled the cost value of a known permutation, thus proving optimality for those instances. Of those, three instances were of dimension $n = 20$ and two had never been previously proved optimal.
- Using the preprocessing technique of Chakrapani and Skorin-Kapov (Chakrapani & Skorin-Kapov, 1994), we were able to improve two of the QAPLP bounds derived from non-preprocessed

Table 17: Instances for which QAPLP is not the best known bound

	lower bounds		
name	best	qaplp	$\frac{\text{best}-\text{qaplp}}{\text{best}}$
nug08	210	204	0.0286
nug12	528	523	0.0095
nug15	1083	1041	0.0388
esc16h	708	704	0.0057
nug20	2394	2182	0.0886
bur26d	3717706	3705831	0.0032
lipa30a	13147	12401	0.0567
nug30	5772	4805	0.1675
tho30	136447	100784	0.2614

data. Lower bounds of 209 and 1046 were produced for `nug08` and `nug15`, respectively. Because of the symmetry of the QAP matrices, the value of 209 can be shifted to 210, a new best known lower bound for `nug08`.

- ADP, as setup for these experiments, uses two stopping criteria: relative improvement of the dual objective function and number of nonzero elements in the factors of the preconditioner. In the 63 instances considered, ADP terminated due to the relative improvement of the dual objective function in 42 runs and because of excessive nonzero elements in the preconditioner in the remaining 21 runs. Increasing the maximum number of conjugate gradient iterations (set at 800 for these experiments) may result in fewer refactorizations, and consequently fewer nonzero elements in the factors of the the preconditioners. This may lead to some instances that terminated because of dense preconditioners to terminate due to relative improvement of the dual objective function, possibly resulting in slightly improved bounds.
- Summed over all runs, the conjugate gradient algorithm accounted for 74.3% of the total running time of ADP. The computation of the preconditioners is done in three steps: reordering of the rows of the A matrix, symbolic factorization, and numeric factorization. These accounted for 1.4%, 0.2%, and 0.3% of the total running time, respectively. The computation of the preconditioners thus accounted for less than 2% of the total running time of ADP.

4 Concluding remarks

We have presented computational testing of the linear programming-based lower bound for the quadratic assignment problem studied by Drezner (Drezner, 1994). To solve the linear programs we use the code first presented by Karmarkar and Ramakrishnan in 1988 at the 13th International Symposium on Mathematical Programming, in Tokyo (Karmarkar & Ramakrishnan, 1988) and further described in (Karmarkar & Ramakrishnan, 1991). The lower bounds produced improved the best known bounds for most of the test problems having QAP dimension $n \leq 30$ from the suite of test problems QAPLIB. In several instances, tight bounds were produced.

We are aware that to replicate our results one requires a solver capable of solving these large scale linear programming problems. Since ADP is restricted for use within AT&T, and the authors are unaware of another linear programming solver that can solve the entire set of 63 linear programming problems, we make available all 63 AMPL (Fourer, Gay, & Kernighan, 1993) models, as well as all 63 feasible dual vectors that correspond to the best QAPLP bounds produced, so that the correctness of the bounds can be verified. Furthermore, we make available detailed iteration summaries for all 63 runs. For each iteration, we indicate iteration number, type of iteration (affine or centering), if refactorization was necessary, and if so, cpu times for reordering, symbolic factorization, and numerical factorization, preconditioner statistics, including cpu time to apply preconditioner, number of dropped nonzero elements from $AA^\top$ matrix and fill-in, conjugate gradient statistics, including conjugate gradient iterations, final residue, final angle between direction and right hand side vector, and cpu time, objective function value, total cpu time for iteration, and total cpu time since start of algorithm.

Much remains to be achieved in terms of proving optimality of quadratic assignment problems. Problems as small as dimension $n = 16$ still challenge researchers at attempts to prove optimality. An interesting research project is to incorporate these LP-based bounds into an exact method, based on branch and bound, to solve quadratic assignment problems, and study the tradeoff between the effect of the improved lower bounds and the more computationally intensive computation of the bounds, as compared with the classical Gilmore-Lawler lower bounds that are commonly used in exact methods. Incoporating interior point methods in a branch and bound algorithm for integer programming has been investigated by Brochers and Mitchell (Brochers & Mitchell, 1992).

Reference

Adams, W., & Johnson, T. (1994). Improved linear programming-based lower bounds for the quadratic assignment problem. In Pardalos, P., & Wolkowicz, H. (Eds.), *Quadratic assignment and related problems*, Vol. 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 43–75. American Mathematical Society.

Assad, A., & Xu, W. (1985). On lower bounds for a class of quadratic $\{0,1\}$ programs. *Operations Research Letters*, *4*, 175–180.

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

Brochers, B., & Mitchell, J. E. (1992). Using an interior point method in a branch and bound algorithm for integer programming. Tech. rep., Rensselaer Polytechnic Institute.

Burkard, R., Karisch, S., & Rendl, F. (1991). QAPLIB – a quadratic assignment problem library. *European Journal of Operational Research*, *55*, 115–119. Updated version – Feb. 1994.

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

Carraresi, P., & Malucelli, F. (1992). A new lower bound for the quadratic assignment problem. *Operations Research*, *40*(Supplement 1), S22–S27.

Chakrapani, J., & Skorin-Kapov, J. (1994). A constructive method for improving lower bounds for a class of quadratic assignment problems. *Operations Research*, *42*, 837–845.

Christofides, N., & Gerrard, M. (1981). A graph theoretic analysis of bounds for the quadratic assignment problem. In Hansen, P. (Ed.), *Studies on graphs and discrete programming*, pp. 61–68. North-Holland.

Drezner, Z. (1994). Lower bounds based on linear programming for the quadratic assignment problem. Tech. rep., Dept. of Management Science, California State University, Fullerton, CA 92634. To appear in *Computational Optimization & Applications*.

Finke, G., Burkard, R., & Rendl, F. (1987). Quadratic assignment problems. *Annals of Discrete Mathematics*, *31*, 61–82.

Fleurent, C., & Ferland, J. (1994). Genetic hybrids for the quadratic assignment problem. In Pardalos, P., & Wolkowicz, H. (Eds.), *Quadratic assignment and related problems*, Vol. 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 173–188. American Mathematical Society.

Fourer, R., Gay, D., & Kernighan, B. (1993). *AMPL – A modeling language for mathematical programming*. The Scientific Press, South San Francisco, CA.

Francis, R., & White, J. (1974). *Facility Layout and Location*. Prentice-Hall, Englewood Cliffs, N.J.

Frieze, A., & Yadegar, J. (1983). On the quadratic assignment problem. *Discrete Applied Mathematics*, *5*, 89–98.

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

Hadley, S., Rendl, F., & Wolkowicz, H. (1990). Bounds for the quadratic assignment problem using continuous optimization techniques. In *Integer Programming and Combinatorial Optimization*, pp. 237–248. University of Waterloo Press.

Hadley, S., Rendl, F., & Wolkowicz, H. (1992a). A new lower bound via projection for the quadratic assignment problem. *Mathematics of Operations Research*, *17*(3), 727–739.

Hadley, S., Rendl, F., & Wolkowicz, H. (1992b). Nonsymmetric quadratic assignment problems and the Hoffman-Wielandt inequality. *Linear Algebra and its Applications*, *58*, 109–124.

Hubert, L. (1987). *Assignment methods in combinatorial data analysis*. Marcel Dekker, Inc., New York, NY 10016.

Karisch, S., & Rendl, F. (1994). Lower bounds for the quadratic assignment problem via triangle decompositions. Tech. rep. 286, CDLDO-40, Technische Universität Graz, Streyrergasse 30, A-8010 Graz, Austria.

Karmarkar, N., & Ramakrishnan, K. (1988). Implementation and computational results of the Karmarkar algorithm for linear programming, using an iterative method for computing projections. Tech. rep., AT&T Bell Laboratories, Murray Hill, NJ.

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

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

Krarup, J., & Pruzan, P. (1978). Computer-aided layout design. *Mathematical Programming Study*, *9*, 75–94.

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

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

Li, Y., Pardalos, P., & Resende, M. (1994b). A greedy randomized adaptive search procedure for the quadratic assignment problem. In Pardalos, P., & Wolkowicz, H. (Eds.), *Quadratic assignment and related problems*, Vol. 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 237–262. American Mathematical Society.

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

McCormick, E. (1970). *Human Factors Engineering*. McGraw-Hill, New York.

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

Nugent, C., Vollmann, T., & Ruml, J. (1969). An experimental comparison of techniques for the assignment of facilities to locations. *Journal of Operations Research*, *16*, 150–173.

Pardalos, P., & Wolkowicz, H. (Eds.). (1994). *Quadratic assignment and related problems*. DIMACS Series on Discrete Mathematics and Theoretical Computer Science. American Mathematical Society.

Skorin-Kapov, J. (1990). Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, *2*(1), 33–45.

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

Thonemann, U.W. and Bölte, A.M. (1994). 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.