

Benchmarking Interior Point LP/QP Solvers

H. D. Mittelmann*

November 26, 1998

Abstract

In this work results of a comparison of five LP codes, BPMPD, HOPDM, LOQO, LIPSOL, and SOPLEX are reported and also of the first three as QP solvers. Since LOQO can solve general NLP problems it is in another class. For LP/QP problems it proves to be robust but it solves certain LP problems somewhat slower due to its limited presolve feature. SOPLEX as the only simplex-based program is highly competitive in general but is beaten by the best IPM codes on certain problems. Among the IPM codes BPMPD stands out while HOPDM has not been perfected as much for the solution of LP/QP problems but rather for use in other contexts requiring its pioneering warmstart feature which is now also available for BPMPD. LIPSOL is the only code in Matlab which has both advantages and disadvantages. It is a pure LP solver and has thus limited applicability compared to the other codes but solves LP problems with an efficiency close to that of BPMPD and HOPDM.

1 Introduction

It is a widely held view that interior point methods (IPM) have matured to a point that a very reliable and efficient solution of LP problems and, more recently, also of convex QP problems, can be achieved. In fact, researchers who have put much effort into this have turned to other aspects, such as generalization to semidefinite programming (SDP), to nonconvex, nonquadratic minimization, to warmstarts and other issues arising from the utilization of their approach in a more general framework of continuous or discrete optimization.

The fact that IPM-LP implementations had reached this level of maturity combined with the fact that a standard input format (MPS) exists and large collections of data files are held in accessible repositories, made it feasible to benchmark codes and make the results available. Ideally, a benchmark run produces results that are reproducible by everyone interested, at least for publicly available codes. Our benchmarking effort [15] has become a frequently cited and visited Web resource. At present, the majority of the codes composed are available free or free for research purposes. In the following a small selection of the codes tested in [15] and a small but informative selection of the test problems used there will be considered. The problem classes are limited to LP and convex QP because very few IPM codes are accessible

*Department of Mathematics, Arizona State University, Tempe, AZ 85287-1804, Tel 602 965-6595, Fax 602 965-0461, e-mail: mittelmann@asu.edu

that can solve more general problems with the exception of SDP problems for which also a number of implementations are available, see [16]. An SDP benchmark was just added to [15].

The following IPM codes will be compared for LP problems: BPMPD, HOPDM, LIPSOL, and LOQO. Links to all codes are on [16]. While several more from [15] could have been included, all but one of the above can also be applied to QP problems which was the main reason for their selection. One code from [15] should be mentioned: PCx. It is available for free with full source and a parallel version is in preparation.

The following is meant to be a concise, self-contained report on the performance of the codes under consideration. Thus, a section on the mathematical background and on the benchmarked codes follows next. The fourth and fifth sections deal with LP, respectively QP, problems, while some conclusions are drawn in the last section.

2 The Primal-Dual Infeasible Interior Point Method for LP and Convex QP

Among various IPM applicable to LP problems, the primal-dual (PD) approach turned out both to be capable of very efficiently solving large problem instances and to permit a rather complete theory including complexity results. This led to entire books being devoted to this class of methods, such as [20] to which reference is made for more in-depth information.

Since we plan to include convex QP problems in the benchmarks and since the inclusion of the quadratic term does not present a major difficulty either in the outline of the methods or in their implementation, we chose to consider the problem

$$\begin{aligned} \min \quad & c^T x + \frac{1}{2} x^T Q x \\ \text{subject to} \quad & Ax = b, x \geq 0 \end{aligned} \tag{1}$$

where $c, x \in \mathbf{R}^n$, $b \in \mathbf{R}^m$, $A \in \mathbf{R}^{m,n}$ of full row rank, $Q \in \mathbf{R}^{n,n}$ symmetric and positive semidefinite (SPSD). The dual of this problem is

$$\begin{aligned} \max \quad & b^T y - \frac{1}{2} x^T Q x \\ \text{subject to} \quad & A^T y + s - Qx = c, s \geq 0 \end{aligned} \tag{2}$$

with dual variables $y \in \mathbf{R}^m$ and dual slacks $s \in \mathbf{R}^n$. The PD formulation combines both problems

$$\begin{aligned} A^T y + s - Qx &= c \\ Ax &= b \\ Xs &= 0 \\ (x, s) &\geq 0, \end{aligned} \tag{3}$$

where $X = \text{diag}(x_i)$.

The nonnegativity constraints on x are replaced by a logarithmic barrier term

$$\begin{aligned} \min \quad & c^T x + \frac{1}{2} x^T Q x - \mu \sum_{j=1}^n \log x_j \\ \text{subject to} \quad & Ax = b. \end{aligned} \tag{4}$$

The first-order necessary optimality conditions for this problem may be written as

$$F(x, y, s) = \left\{ \begin{array}{c} A^T y + s - Qx - c \\ Ax - b \\ XSe - \mu e \end{array} \right\} = 0 \quad (5)$$

where $S = \mu X^{-1}$, $e = (1, \dots, 1)^T \in \mathbf{R}^n$. This nonlinear system may be solved using Newton's method

$$\begin{aligned} J_F(z^{(k)})\Delta z^{(k)} &= -F(z^{(k)}) \\ z^{(k+1)} &= z^{(k)} + \alpha_k \Delta z^{(k)} \end{aligned} \quad (6)$$

with $z^{(k)} = (x^{(k)}, y^{(k)}, s^{(k)})$, $\Delta z^{(k)} = (\Delta x^{(k)}, \Delta y^{(k)}, \Delta s^{(k)})$, and the Jacobian

$$J_F(z^{(k)}) = \begin{pmatrix} -Q & A^T & I \\ A & 0 & 0 \\ S^{(k)} & 0 & X^{(k)} \end{pmatrix}. \quad (7)$$

The solutions of $F = 0$ are called *analytic centers*. They are parametrized by $\mu \geq 0$ and $\mu = 0$ characterizes the solution $z^* = (x^*, y^*, s^*)$. Again, reference is made to [20] and the bibliography below for more detailed information. A few more definitions, however, need to be quoted. The residuals f_i , $i = 1, 2$, of $F = (f_1, f_2, f_3)$ are called *dual* and *primary infeasibilities* while f_3 is the *complementarity gap*. Only f_3 is nonlinear, namely quadratic, in the variables of the problem. In the solution process one faces the following major tasks

1. find initial guess $z^{(0)}$
2. choose initial parameter value $\mu > 0$
3. solve the linear system in (6)
4. determine the step length $\alpha_k \geq 0$
5. decrease μ towards 0

In the following we will concentrate on the third and fourth issues.

Eliminating $\Delta s^{(k)}$ in the linear system (6) one with the following matrix

$$\begin{pmatrix} -Q - (X^{(k)})^{-1}S^{(k)} & A^T \\ A & 0 \end{pmatrix} \quad (8)$$

is obtained. This is the *augmented matrix* and solving at this level of reduction is called the *augmented system* (AS) approach. A further reduction is possible. Denoting $D^{(k)} = Q + (X^{(k)})^{-1}S^{(k)}$, an SPD matrix which is diagonal for the LP problem, an elimination of $\Delta y^{(k)}$ leads to a system for $\Delta x^{(k)}$ only with matrix

$$A(D^{(k)})^{-1}A^T. \quad (9)$$

This is called the *normal equation* (NE) approach.

All the codes under consideration initially used the NE approach. Due to the fact that the matrix is SPD a modified Cholesky decomposition is possible. Trying to exploit the sparsity of the matrix A , this still requires specialized methods. More recently, for LP and certainly for QP for which D^{-1} is not easily computed, the AS approach is preferred. Its matrix (8), however, is symmetric indefinite. A classical direct algorithm for such cases is the Bunch-Parlett factorization [2].

The other major issue addressed is that of the steplength choice in Newton's method (6). The most successful method initially proposed for the practical solution of the nonlinear system $F(z) = 0$ is Mehrotra's predictor-corrector method [8]. The *predictor step* uses the pure Newton direction obtained above for $\mu = 0$, also called an *affine-scaling* direction. It is thus based on a first order approximation through the Newton linearization of (5). A *centering term* μ is then determined adaptively based essentially on the achievable reduction of the complementarity gap when moving along the affine-scaling direction, in other words on how well the Newton linearization approximates the nonlinear system. Finally, a *corrector iteration* for this centering term is computed using the same system matrix, respectively its factorization, as in the first step. This amounts to a *quadratic* approximation of the nonlinear system and, in a slightly modified form, was implemented in the initial versions of most PD IPM codes. Practical algorithms all use different stepsizes for the primal and dual components, so α_k above is not a scalar but should be taken as a scaling matrix.

3 The Benchmarked Codes

In this section a short summary of the main features will be given for each of the codes BPMPD, HOPDM, LIPSOL, and LOQO. References will be given for more detailed descriptions, see also contributions in this volume. The versions tested were BPMPD-2.21 (2/98), HOPDM-2.30 (8/98), LOQO-4.01 (10/98). Unfortunately, no commercial software was available for a comparison.

The only code that only solves LP problems is LIPSOL. It is written in Matlab and the current version for Matlab 4 was completed in 1995 [22]. It was recently ported to Matlab 5. A more detailed description appeared in 1996 in [23]. LIPSOL uses the NE approach and the Ng-Peyton sparse Cholesky factorization together with the minimum degree (MD) ordering of Liu in the form of MEX files from f77 subroutines provided with the code. LIPSOL in slightly modified form follows [8] concerning issues such as starting point selection, determination of centering parameter and steplength choice. It has a dense column treatment which is enhanced by preconditioned cg iterations in case the application of the Sherman-Morrison formula seems unsatisfactory, see [23].

Another main feature described in detail in [23] is a stabilization procedure for the sparse Cholesky factorization in case of near singular matrices as they typically arise in the end phase of the solution process. LIPSOL solved all feasible NETLIB problems and the results were given in [23].

The code BPMPD and HOPDM both solve convex QP problems. As the name indicates, HOPDM was the first to use higher order primal dual methods. This refers to adaptively determined *multiple centrality corrections*. Immediately following the preprocessing/reordering stage heuristically the number of such corrections is determined. The method uses the Mehro-

tra second order method as predictor. Instead of following higher-order approximations of the central path, first the stepsizes in both primal and dual spaces are enlarged, in general leading outside the feasible region. Then, a corrector direction is determined which points to a *target* point, not on the central path but only in a relatively large neighborhood of it. Each such correction involves one more solve with the factored matrix and one ratio test. It is being stressed in [4] that the method primarily aims at improving centrality and not at reducing the complementarity gap. It is shown with difficult NETLIB examples that one (two) centrality corrections lead to a 15%–25% (20%–35%) reduction in the number of iterations.

The LP version of HOPDM uses the NE approach while the QP version [1] naturally solves the AS. The key feature of the QP version is an adaptive regularization of the augmented matrix combined with a standard Cholesky decomposition. As in [17], *quasidefinite* matrices of the form

$$\begin{bmatrix} -E & A^T \\ A & F \end{bmatrix}$$

with E, F both SPD are considered. No permutations are performed as in [17], but only diagonal pivots are chosen in the quasidefinite matrix

$$\begin{bmatrix} -Q - X^{-1}S & A^T \\ A & 0 \end{bmatrix} + \begin{bmatrix} -R_p & 0 \\ 0 & R_d \end{bmatrix}$$

where R_p, R_d are nonnegative diagonal regularization matrices determined from the size of the pivots encountered during the factorization. Other elements of the LP version including the multiple centrality corrections were taken into the QP code with minor changes. Detailed test results are reported in [1] including for some difficult large QP test cases from the CUTE collection and randomly generated examples, both taken from [15].

While the basic elements of HOPDM were available several years ago and not many major changes were necessary for its QP extension, the author of the package has concentrated on utilizing the code in other contexts [3] for which a warmstart feature [5] is a crucial ingredient.

The code BPMPD has undergone a constant development from its beginnings as LP code until today when it is available for convex QP problems. The author has documented this development in a large collection of papers. It is the only code that solves all the feasible benchmark problems in [15] and it does that overall with a high efficiency. While LIPSOL and HOPDM have a presolve feature and LOQO deliberately has practically none, BPMPD has a relatively elaborate first stage including an effective elimination of free variables [9]. It uses a heuristic to decide between the NE and the AS approaches [6] for both LP and QP problems.

The factorization does not utilize work by others but was developed in [10] and the sparsity handling for QP problems in [11]. Degeneracy is handled by a special scaling-independent pivot tolerance [13]. The ordering algorithm is of the inexact *minimum local fill-in* type [12] and in general slower than MD reordering. The orderings produced, however, often allow more efficient solves. Finally, the steplength choice was improved for quadratic problems [14] by considering the problem of minimizing primal and dual infeasibilities as a multicriteria optimization problem.

The code LOQO is not available as source. From its beginnings as an LP solver, LOQO has progressed to a nonconvex NLP program [18] and thus has capabilities which, although

tested to some degree in [15], go beyond those of the other codes compared. We will, therefore, limit consideration of LOQO to convex QP problems. The original QP version of LOQO is described in [19] and is not identical to the current NLP version. The modifications for the latter concern especially the choice of the initial point, of the search direction and the steplength, while other issues such as matrix reordering and stopping criteria are handled the same way. As mentioned above, LOQO's author was the first to consider modified Cholesky decomposition of quasidefinite matrices [17]. No Bunch-Parlett type decomposition is needed by LOQO.

no.	problem	m	n	nz
1	baxter	23185	14947	90697
2	dano3mip	3202	13873	79655
3	dbir1	8401	25016	1048201
4	df001	6071	12230	35632
5	fxm3.16	36027	62779	361489
6	gosh	3395	10025	88818
7	klein3	993	88	12106
8	nsct1	11516	11304	644874
9	rat7a	3136	9408	268908
10	rlfprim	57422	8048	264483
11	route	26894	23923	187686
12	scagr7.2r.864	23340	34579	92487
13	seymour	4827	1255	33432
14	watson.10.1536	119782	230361	684989
15	world	29659	31434	130073

Table 1: Data of the LP benchmark problems

As in papers on the other codes, results on a number of test problems, typically from the NETLIB collection, serve to support the choice of certain algorithmic techniques. This led to a criticism voiced in [20, end of Ch. 10] that this collection may have had too much influence on the development of IPM. In the case of LOQO, a clever priority-controlled pivot choice is partly justified this way and also compared to the MD reordering. This comparison is not based on fill-in but on the number of arithmetic operations required for the factorization. It is argued and demonstrated with test results that the work is comparable but that the priority-based reordering often considerably improves numerical stability.

Since always only the latest version of LOQO is available which recently has been considerably generalized over the QP version, it is difficult to go into more detail concerning its treatment of or performance on convex QP problems alone. If LOQO recognizes a problem as QP it chooses certain values for its key parameters automatically and in general solves these problems quite efficiently. LOQO has an integrated AMPL interface, is available with GAMS, and for QP problems there is a Matlab interface provided.

no.	problem	BPMPD	HOPDM	LIPSOL	LOQO	SOPLEX
1	baxter	41	4205	4487	151	70
2	dano3mip	1241	626	1500s	1719	825
3	dbir1	431	1877	m	5810	435
4	df001	562	2541a	3859i	6582	672
5	fxm3.16	122	147	199	238	1490
6	gosh	14	51	40	208	108
7	klein3	1	2	3092	5	2
8	nsct1	715	6745	m	27353	193
9	rat7a	159	224	644	3759	3200
10	rlfprim	104	771	m	1843	59
11	route	48	355	323	104	125
12	scagr7.2r.864	38	45	f	352	1076
13	seymour	11	14	4623	27	25
14	watson.10.1536	685	1297	m	1911	19123
15	world	270	299	899	641	8152

Table 2: CPU times for the LP test problem

4 The LP Benchmark

In this section the codes BPMPD, HOPDM, LIPSOL, LOQO, and SOPLEX [21] are compared on a sample of LP problems chosen to show differences between codes. It should be stressed at the outset that all the codes tested, as well as a few not included here but in [15], are capable of solving the vast majorities of available LP test problems very well. Since at least some of these problems were taken from *real-life* applications, it can be said that a user in need of a stable and efficient LP solver can choose any of these codes. Typically, if there is a problem such as reduced accuracy in the solution, the codes will inform the user of this fact. Still, there are differences resulting from the specific algorithms chosen, their implementations, and even the language of the programs.

All codes accept input in MPS format, so only files in this format were chosen for the tests. Sources are given in [15]. All tests were done on the same platform, a Sun Ultra Sparc (300 MHz, Solaris 2.6, 512 MB). For LOQO the binary provided was used, the computational core of LIPSOL was compiled with the native f77 compiler and the compile options given in the makefile. It was run under Matlab 5. SOPLEX was compiled with gcc-2.7 (it did not compile with gcc-2.8) and the options given in the makefile. BDMPD(C) and HOPDM(f77) were compiled with the native compilers and the *fast* option. All codes were run with default options.

Table 1 lists some key data of the testset. **nz** denotes the number of nonzeros in A . The problems 2–6, 8, 9, 14, 15 should be considered as essentially having no dense columns. The problems 6, 7 are infeasible and problem 4 is the only feasible problem from the NETLIB collection. It was chosen because for many years when various simplex-based codes were tested with the set this was one of the more challenging problems. If one assumes that by now there should be no problems left in solving it, a glance at Table 2 which lists the total

CPU times including for the input of the MPS-files, reveals that this is not quite true. For one, the simplex code solves it in nearly the same time as the best IPM code and two of the latter do not terminate with a sufficiently accurate solution while the remaining IPM code takes about 15 times as long as the successful solvers.

no.	problem	BPMPD				HOPDM	LOQO
		m	n	$nz(A)$	$nz(L)$	$nz(L)$	$nz(L)$
1	baxter	19231	11105	95971	116992	6265025	338535
2	dano3mip	2636	13308	79655	1702921	1161037	1153648
3	dbir1	7147	25012	1058605	1703253	3654470	3634441
4	df001	3778	9949	35632	1074236	1516980	1603134
5	fxm3.16	32690	58162	370839	569758	537139	841384
6	gosh	2787	9339	97231	179011	196549	349373
7	klein3	993	88	12107	12112	16919	15926
8	nsct1	7695	11303	656259	2554411	5559513	5625692
9	rat7a	2152	6772	268908	936691	1030035	3527073
10	rlfprim	53406	4132	265927	270631	3079402	3016089
11	route	20779	23923	187686	216114	3078015	643848
12	scagr7.2r.864	12966	23341	108042	81077	151278	151205
13	seymour	4827	1172	33549	34702	162909	138766
14	watson.10.1536	69360	179939	1052028	1651451	1774745	3178790
15	world	27350	28866	198793	991805	1038129	1150773

Table 3: Sizes of LP problems after presolves for BPMPD and sizes of L

The notation in Table 2 is the following. A failure, and there is only one, is denoted by **f**. An **a** after a time denotes that the algorithm terminated but reported that only a rather moderate accuracy had been reached. A letter **s** denotes that the program stalls without termination but is close to a solution, while an **i** indicates that it erroneously reported the problem to be infeasible although it was close to a solution. An **m** indicates that sufficient memory could not be allocated. One of the reasons that LOQO frequently takes longer than most other programs is the fact that only a minimal presolve is performed. The author expects the user of the program to supply data which do not need extensive preprocessing. Another inherent problem is the use of the Matlab environment by LIPSOL. It certainly makes the code attractive to users of Matlab; however, the generous memory allocation of Matlab prevented the code from fully participating in the benchmark. While all the other codes used only a fraction of the available memory in cases 3, 8, 10, 14, LIPSOL did not run. This is a drawback for the solution of large problems.

The only simplex-based code SOPLEX available with full source for research purposes and certainly one of the best such implementations has the lowest or second-lowest time in 7 of the 15 test examples. This may be a surprise to some, but there are clearly test problems, such as 14, 15, for which SOPLEX is slower by a factor of 10–30. Other such cases are the nug* problems, see [15]. There are no cases, both here and in [15] for which SOPLEX is faster by a similar factor than the best IPM codes. The fact that IPM codes may have more problems than simplex-based codes in detecting infeasibilities is confirmed

no.	problem	order	no_p	no_c	no_pc
1	baxter	1	50	39	49
2	dano3mip	6	1820	1514	2284
3	dbir1	5	727	474	806
4	df001	6	832	733	1050
5	fxm3.16	1	369	122	370
6	gosh	3	21	14	23
7	klein3	1	1	1	1
8	nsct1	5	3424	844	3367
9	rat7a	5	435	192	486
10	rlfprim	4	557	113	437
11	route	1	88	51	89
12	scagr7.2r.864	1	46	39	44
13	seymour	3	13	12	12
14	watson.10.1536	1	897	685	898
15	world	3	957	303	848

Table 4: Higher order correction and times without certain features for BPMPD

by the results for problems 6, 7. For the problems with dense columns 1, 7, 10–13 SOPLEX does quite well compared to the IPM codes with the exception of 12 which has superdense columns. LIPSOL uses only the NE approach which puts it at a disadvantage compared to the other IPM codes especially when dense columns are present. LIPSOL needs excessive time to solve problem 13.

It seems desirable to obtain more detailed information about the performance of the algorithms and the reasons for differences among them. All algorithms have a sizeable number of tuning parameters, switches, etc., through which the user can influence the solution process. In fact, there are so many for some codes that it is impossible to try comparing not only the default choices, typically found by the author after careful and extensive testing, but any of the combinations possible. However, there are certain key features whose effect is of particular interest and in the following an attempt is made to shed some light on the question of differences in performance. Two key ingredients of the IP algorithms as implemented especially in BPMPD and HOPDM are *presolves* and *higher-order centrality corrections*. Table 3 lists the sizes of the problems after BPMPD’s presolve and the size of the augmented matrix L for BPMPD, HOPDM and LOQO, the latter two performed with their presolves.

In Tables 4 and 5 are listed the order of the centrality corrections and the solution times without presolve (no_p), without higher-order centrality corrections (no_c) and without either (no_pc). For BPMPD the presolve often improves solution times substantially while the higher-order corrections after a presolve yield a sizeable reduction in only about a third of the cases. For HOPDM presolves are at least as important while the higher-order corrections in general lead to an improvement but occasionally solution times would be faster without them. HOPDM frequently but not always solves with an efficiency comparable to that of BPMPD. Not only the latter as mentioned above but also the former has a heuristic, in

no.	problem	order	no_p	no_c	no_pc	NE/AS
1.	baxter	10	4784	5048	6050	57
2	dano3mip	10	647	852	957	1218
3	dbir1	9	2927	2036	3634	1211
4	df001	10	907a	1291a	1230a	*
5	fxm3.16	2	192a	144	1929	156
6	gosh	4	40	31	50	57
7	klein3	3	1	2	1	35
8	nsct1	12	8844	7836	11506	5135
9	rat7a	9	2003	294	3594	365
10	rlfprim	8	1048	895	983	157
11	route	5	438	344	34592	92
12	scagr7.2r.864	2	57	48	58	9204
13	seymour	5	14	15	13	3101
14	watson.10.1536	2	2250	1283	2063	1303
15	world	4	620a	312	718a	388

Table 5: Higher order correction and times without certain features for HOPDM

this case based on the relative density of the columns of A , to choose between the NE and the AS approach. It appears that this strategy could be improved for HOPDM while no such need becomes apparent for BPMPD. The problems that HOPDM solves using the NE factorization are 1, 4, 5, 6, 9, 11, and 15. The last column of Table 5 lists the times obtained when in all cases the opposite factorization method was forced. Significant improvements are possible, particularly for problems 1, 10, 11. BPMPD was choosing the augmented system factorization in cases 1, 7, 10, 11, 12, and 13 while LOQO applies a *dual ordering* in cases 7, 9, 10, 12, and 13.

With an improved heuristic for the selection of NE/AS HOPDM's performance may be even more comparable to that of BPMPD which proved to be the most robust and efficient code among those tested. Again, for a comparison of these and other codes on a larger selection of test problems, reference is made to [15].

5 The QP benchmark

In this section the codes BPMPD, HOPDM, and LOQO will be compared on a sample of convex QP problems. Table 6 lists the data of the 14 problems considered. Here, $\mathbf{nz}(\mathbf{Q})$ denotes the number of nonzero off-diagonal elements in one half of Q . The first five were run on an HP9000-K260 workstation with R8000 chip and 256MB memory, while the others ran on a DEC-Alpha 500-500 workstation with 500 MHz chip and 512 MB memory. The problems "control*" are taken from Example 5.1 in [7] while the others were already used in [15] where sources are given. The testcases QP* and SQP* are randomly generated including a randomly chosen sparse structure. As the results show all codes perform very comparably on such problems. The "control*" problems on the other hand have a simple,

namely diagonal Q matrix and a discretized PDE as constraint. For these problems the times for BPMPD and LOQO are in a similar relationship as for LP problems while HOPDM shows less stable behavior. A **o** indicates that these are times for LOQO-3.10. LOQO-4.01 exceeded the available memory (512MB) and paging would have been necessary. For the case *control 1-100* the maximum iterations (120) are reached. Increasing this limit at iteration 297 a suboptimal solution is reached while for *control 1-200* even 500 iterations are not sufficient. The last 6 problems are from the CUTE collection but with greatly increased dimensions. For these, supposedly more “real-life” problems HOPDM’s times are mostly of the same order as those of BPMPD while LOQO solves all problems but, as for LP, considerably slower. In the case of CVXQP3 it is again HOPDM that does not reach full accuracy. Again, it is of interest how the higher-order centrality correction affects performance. Table 8 lists the orders used and times without such corrections for BPMPD and HOPDM for the last 9 problems. As a reference code MINOS-5.5 employing an active-set method was run on the QP problems. Because of excessive memory and CPU time requirements the problems marked by an **r** were solved with reduced dimensions (m/n): AUG3DC: 3376/12108, AUG3DCQP: 3376/12108, AUG3DQP: 4097/14547). CVXQP1 and CVXQP3 could not be run without changes to the code.

problem	m	n	$nz(A)$	$nz(Q)$
QP500-2	1100	500	51788	89071
QP1000-1	700	1000	15753	68762
control 1-50	2401	2597	12005	0
control 1-100	9801	10197	49005	0
control 1-200	39601	40397	198005	0
SQP2500-1	2000	2500	52321	738051
SQP2500-2	2000	2500	52319	14345
SQP2500-3	4500	2500	115073	738051
AUG3DC	15625	52428	112981	0
AUG3DCQP	27000	89013	181320	0
AUG3DQP	27000	89013	181320	0
CVXQP1	7500	15000	22497	44981
CVXQP2	3750	15000	11249	44981
CVXQP3	11250	15000	33745	44981

Table 6: Data of the QP benchmark problems

6 Conclusions

A number of codes were tested as LP solvers and a subset as a QP solver. While all codes show considerable robustness and efficiency there are also clear differences in either of these areas. The code LOQO with its capability of solving general NLP problems far exceeds the applicability of the other codes. It behaves solid but not as efficient in the solution of both LP and QP problems. An interesting result of the LP test is that the only simplex-based

problem	BPMPD	HOPDM	LOQO	MINOS
QP500-2	27	26	18	5850
QP1000-1	112	106	112	5247
control 1-50	3	9	8	67
control 1-100	21	*	84	1207
control 1-200	155	*	1277	*
SQP2500-1	956	1432	1215	396
SQP2500-2	2068	1748	2175	413
SQP2500-3	1003	1406	1673	18733
AUG3DC	53	241	424	67063r
AUG3DCQP	991	1939	8813	34318r
AUG3DQP	595	2175	5489	21296r
CVXQP1	1603	3381	26283o	-
CVXQP2	963	2729	30337	4687
CVXQP3	5173	*	73140o	-

Table 7: CPU times for QP test problems

code is very competitive and what clearly seems to be called for is a comparison in a context that requires warmstarts between this code and the two IPM programs with such a feature, BPMPD and HOPDM. Overall one code had an edge in both efficiency and robustness for both LP and QP problems, namely BPMPD with HOPDM a close second followed by LIPSOL and LOQO, the former affected a bit by its need to call Matlab and also somewhat less robust. It needs to be stressed again that LOQO solves *nonlinear, nonconvex* problems and does this remarkably well as benchmarks in [15] have shown. It is the most versatile and for that a very robust code.

References

- [1] A. Altman and J. Gondzio, *Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization*, Logilab Technical Report 1998.6, HEC Geneva, Department of Management Sciences, University of Geneva, Geneva, Switzerland, 1998.
- [2] J.R. Bunch and B.N. Parlett, *Direct methods for solving symmetric indefinite systems of linear equations*, SIAM Journal on Numerical Analysis 8 (1971), 639–655.
- [3] J.L. Goffin, J. Gondzio, R. Sarkissian and J.P. Vial, *Solving nonlinear multi-commodity flow problems by the analytic center cutting plane method*, Interior point methods in theory and practice, Math. Programming 76 (1997), 131–154.
- [4] J. Gondzio, *Multiple centrality corrections in a primal-dual method for linear programming*, Computational Optimization and Applications 6 (1996), 137–156.

problem	BPMPD		HOPDM	
	order	no_c	order	no_c
SQP2500-1	7	1007	11	1506
SQP2500-2	7	2258	11	2188
SQP2500-3	7	1181	11	1853
AUG3DC	5	60	8	241
AUG3DCQP	6	1022	10	2541
AUG3DQP	6	674	10	2275
CVXQP1	7	1842	11	3628
CVXQP2	7	418	11	2719
CVXQP3	7	5466	11	*

Table 8: Higher-order correction and times without them

- [5] J. Gondzio, *Warm start of the primal dual method applied in the cutting plane scheme*, Logilab Technical Report 96.3, University of Geneva, to appear in Math. Programming.
- [6] I. Maros and Cs. Mészáros, *The role of the augmented system in interior point methods*, European Journal of Operations Research 107 (1998), 720–736.
- [7] H. Maurer and H.D. Mittelmann, *Optimization techniques for solving elliptic control problems with control and state constraints. Part 1: Boundary Control*, to appear in Computational Optimization and Applications.
- [8] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM Journal on Optimization 2 (1992), 575–601.
- [9] Cs. Mészáros, *On free variables in interior point methods*, Optimization Methods and Software 9 (1998), 121–139.
- [10] Cs. Mészáros, *Fast Cholesky factorization for interior point methods of linear programming*, Computers & Mathematics with Applications 31 (1996), 49–54.
- [11] Cs. Mészáros, *On the sparsity issues of interior point methods for quadratic programming*, Working paper WP 98-4, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, 1998.
- [12] Cs. Mészáros, *The “inexact” minimum local fill-in ordering algorithm*, Working paper WP 95-7, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, 1995.
- [13] Cs. Mészáros, *On a property of the Cholesky factorization and its consequences for interior point methods*, Working paper WP 98-7, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, 1998.
- [14] Cs. Mészáros, *Steplengths in infeasible primal-dual interior point algorithms of quadratic programming*, DOC 97/7, Imperial College, Department of Computing, London, 1997.

- [15] H.D. Mittelmann and P. Spellucci, *Benchmarks for optimization software*, World Wide Web, <http://plato.la.asu.edu/bench.html>, 1998.
- [16] H.D. Mittelmann and P. Spellucci, *Decision tree for optimization software*, World Wide Web, <http://plato.la.asu.edu/guide.html>, 1998.
- [17] R.J. Vanderbei and T.J. Carpenter, *Symmetric indefinite systems for interior point methods*, *Mathematical Programming* 58 (1993), 1–32.
- [18] R.J. Vanderbei and D.F. Shanno, *An interior point algorithm for nonconvex nonlinear programming*, Dept. of Statistics and Operation Research, SOR-97-21, Princeton University, Princeton, NJ, 1997.
- [19] R.J. Vanderbei, *LOQO: An interior point code for quadratic programming*, Dept. of Statistics and Operations Research, SOR-94-15, Princeton University, Princeton, NJ, 1994 (revised 10/6/1998).
- [20] S. Wright, *Primal-dual interior point methods*, SIAM, Philadelphia, 1997.
- [21] R. Wunderling, *Paralleler und Objektorientierter Simplex*, TR 96-09, Konrad Zuse Center for Scientific Computing, Berlin, Germany, 1996.
- [22] Y. Zhang, *User's guide to LIPSOL*, Technical Report TR 95-19, Dept. of Math. and Statistics, Univ. of Maryland, Baltimore County, Baltimore, MD, 1995.
- [23] Y. Zhang, *Solving large-scale linear programs by interior point methods under the matlab environment*, Technical Report TR96-01, Dept. of Math. and Statistics, University of Maryland Baltimore County, Baltimore, MD, 1996.