On the factorization of simplex basis matrices

R. LUCE, J. DUINTJER TEBBENS, J. LIESEN and R. NABBEN
Technical University of Berlin
M. GRÖTSCHEL and T. KOCH
Zuse Institute Berlin
and
O. SCHENK
University of Basel

In the simplex algorithm, solving linear systems with the basis matrix and its transpose accounts for a large part of the total computation time. The most widely used solution technique is sparse LU factorization, paired with an updating scheme that allows to use the factors over several iterations. Clearly, small number of fill-in elements in the LU factors is critical for the overall performance.

Using a wide range of LPs we show numerically that after a simple permutation the non-triangular part of the basis matrix is so small, that the whole matrix can be factorized with (relative) fill-in close to the optimum. This permutation has been exploited by simplex practitioners for many years. But to our knowledge no systematic numerical study has been published that demonstrates the effective reduction to a surprisingly small non-triangular problem, even for large scale LPs.

For the factorization of the non-triangular part most existing simplex codes use some variant of dynamic Markowitz pivoting, which originated in the late 1950s. We also show numerically that, in terms of fill-in and in the simplex context, dynamic Markowitz is quite consistently superior to other, more recently developed techniques.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Linear systems (direct and iterative methods)*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: direct factorization methods, dynamic Markowitz pivoting, large and sparse linear systems, simplex-based LP solvers

---

The work of J. Duintjer Tebbens was supported by the MATHEON DFG Research Center, Berlin. The work of R. Luce and J. Liesen was supported by the Emmy Noether-Prgramm of the Deutsche Forschungsgemeinschaft.
Author's addresses: R. Luce, J. Duintjer Tebbens, J. Liesen, R. Nabben, Institute of Mathematics, Technical University of Berlin, Straße des 17. Juni 136, 10623 Berlin, Germany (`[luce, duintjer, liesen, nabben]@math.tu-berlin.de`); M. Grötschel, T. Koch, Zuse Institute Berlin (ZIB), Takustraße 7,14195 Berlin, Germany (`[groetschel,koch]@zib.de`); O. Schenk, Department of Computer Science, University of Basel, Klingelbergstrasse 50, 4056 Basel, Switzerland (`olaf.schenk@unibas.ch`)

1. INTRODUCTION

Numerous applications in optimization require solving large-scale linear programs (LPs). For example, large-scale LPs arise as subproblems in mixed-integer programs (MIPs). The most widely used LP solution method in this context is the simplex algorithm, which is considered one of the most important algorithmic developments of the 20th century [Cipra 2000]. Each step of this algorithm requires solving (at least) two large, very sparse and nonsymmetric linear systems of equations, one with the current basis matrix and one with its transpose. In most existing computer codes, these two solves are performed directly, based on an LU factorization of the basis matrix. For this factorization, dynamic Markowitz pivoting, originally described in [Markowitz 1957], is used, usually in some variant of the implementation described by Suhl and Suhl [1990]. Comprehensive information on computational aspects of the simplex algorithm is given in [Maros 2003; Koberstein 2005; Wunderling 1996].

Interior-point methods offer an alternative to the simplex algorithm for the solution of large-scale LPs. At the core of these methods highly ill-conditioned symmetric saddle point matrices due to the Karush-Kuhn-Tucker optimality conditions have to be solved. In recent years a large amount of work has been devoted to solution methods for such systems [Gould et al. 2007]. For matrices arising in interior-point methods fast factorization algorithms have led to significant performance advances [Duff and Pralet 2005; Schenk and Gärtner 2006; Schenk et al. 2007]. Another example where recent developments in sparse factorization techniques result in improved LP solution algorithms is provided by Davis and Hager [2008], who focus on computational techniques for the dual active set algorithm.

Motivated by this progress in sparse factorization for interior-point and dual-active set methods, we investigate in this paper the potential of using modern sparse direct factorization techniques (for nonsymmetric systems) in the linear algebra kernel of simplex solvers. We focus on the factorization of the so-called basis matrices. We do not consider updating strategies for the factorization and we do not consider the sparse triangular solves either. Our extensive experimental study shows that modern methods do not offer a competitive alternative to the existing LU factorization based on Markowitz pivoting. This is quite surprising, given that modern linear algebra techniques have been very successful for other optimization methods and applications areas.

The paper is organized as follows. After a short summary of our results in the following section, we give in section 3.1 a “working definition” of the LP problem and the basis matrix, and a high level description of the simplex algorithm. In section 3.2 we discuss structural features of LP basis matrices. Then, in section 4, we present the results of our benchmarks in detail. Further numerical results are given by [Luce 2007].

2. SUMMARY OF THE RESULTS OF OUR BENCHMARK

A major slice of the computational time within the simplex algorithm is spent for the solves with the factors of the basis matrix. The fill-in in these factors is of major importance for the overall performance of the simplex algorithm. In our experiments we therefore concentrate on the fill-in in the factors rather than on factorization times, which usually account for less than 5% of the overall simplex runtime. Our main experimental results have been performed with the LP code SoPlex[1] (sequential object-oriented simplex, v. 1.3.0, see [Wunderling 1996]). We have chosen SoPlex because it implements the most widely used linear algebra techniques and allows easy source code instrumentation for recording all relevant data. It would have been interesting to aquire the same data from a commercial simplex solver, but to our knowledge no such code offers appropriate API ("application programming interface") routines to access the data needed for this benchmark. Our large set of linear programming problems includes models from NETLIB[2], MIPLIB 2003 [Achterberg et al. 2006], the Mittelmann Benchmark LPs[3], and some large-scale LPs provided to us by the Zuse Institute Berlin (ZIB). Our main results can be summarized as follows:

(1) The LP basis matrices typically admit an LU factorization with a relative fill-in close to optimal. The reason for this is that an overwhelming part of the basis matrix can be permuted to triangular form. Only a very small remaining part, called the *nucleus*, still needs to be factorized, especially for large scale problems. The potential fill-in in the basis factorization, since restricted to the nucleus, is very small.

(2) In the factorization of the small nuclei, the dynamic Markowitz pivoting strategy as implemented in SoPlex typically produces a fill-in which is as good as or even better than a selection of state-of-the-art LU codes, namely Pardiso [Schenk and Gärtner 2004; 2006; Schenk et al. 2000], Umfpack [Davis 2004a; 2004b] and Wsmp [Gupta 2002a; 2002b].

Concerning our result stated in item (1), it is clear that triangular parts of the basis matrix have been exploited ever since computational aspects of the simplex algorithm have been explored (see [Orchard-Hays 1968] and references within or [Tomlin 1972] for a presentation in the context of LU factorizations). Hence, (1) can be considered as "folklore wisdom" among simplex practitioners. But, to our knowledge, in recent years no extensive benchmark has been performed that provides empirical evidence that this property is in fact the reason why LU factorizations of large-scale simplex basis matrices can be obtained at very low computational cost. In this paper we provide such a benchmark which takes into account many real world large-scale LP instances.

Large parts of the LP basis matrices typically can be triangulated by means of permutations, and thus the computation of LU factorizations of these matrices with small and even close to optimal fill-in can be considered a rather simple problem from the numerical point of view. The simplicity of this task becomes even clearer when noticing that the permutations that perform the triangulations can be found simply by successively moving column- and row-singletons to the front (see section 3.2 for details).

[1] `http://soplex.zib.de/`
[2] `http://www.netlib.org/`
[3] `http://plato.asu.edu/bench.html`

3. MATHEMATICAL BACKGROUND

3.1 The (revised) simplex algorithm

In this section we give a "working definition" of the LP problem and basis matrices. We describe the dual simplex algorithm on a conceptual level, focusing only on the steps relevant for our presentation. Our goal is to expose the structure of the basis matrices as simply as possible. There is no loss of generality in comparison with other formulations of the LP problem, or other formulations of the simplex algorithm.[4]

Definition 3.1 *Linear Program (LP).* Let $A \in \mathbb{R}^{m\times n}$ be the constraint matrix, $b \in \mathbb{R}^m$ the right hand side vector, $c \in \mathbb{R}^n$ the cost vector and $s \in \mathbb{R}^m$ the vector of slack variables. The linear programming problem is to find a solution $\binom{x}{s} \in \mathbb{R}^{n+m}$ to the following optimization problem:

$$\begin{array}{lrcl} \max & c^T x & & \\ \text{s.t.} & (A, I_m)\binom{x}{s} & = & b \\ & x, s & \geq & 0 \end{array} \tag{1}$$

The matrix $(A, I_m) \in \mathbb{R}^{m\times(n+m)}$ is called the extended constraint matrix.

In a typical LP we may expect $m \leq n$. The matrix A is usually very sparse, meaning that only a few entries in each row and column are nonzero. For simplicity, we assume that (1) is neither infeasible nor unbounded. Note that the treatment of these cases is an algorithmic aspect of the simplex algorithm and has no impact on our discussion of the solution of linear systems.

Definition 3.2 *Basis.* Consider any partitioning of the set $\{1, 2, \ldots, n+m\}$ into two disjoint subsets $\mathcal{B}$ and $\mathcal{N}$, i.e., $\mathcal{B} \cap \mathcal{N} = \emptyset$ and $\mathcal{B} \cup \mathcal{N} = \{1, \ldots, m+n\}$, with $|\mathcal{B}| = m$. If the matrix $B = (A, I_m)_{\bullet\mathcal{B}} \in \mathbb{R}^{m\times m}$ is nonsingular, we call B the basis matrix corresponding to the set of basic column indices $\mathcal{B}$. The set $\mathcal{N}$ is called the set of non-basic column indices.

(For a matrix Z and a set of its column indices $\mathcal{I}$, the notation $Z_{\bullet\mathcal{I}}$ refers to the submatrix of A induced by all the rows of A and by all the columns subscripted by $\mathcal{I}$.)

The dual simplex algorithm is an iterative procedure for solving (1). It computes a sequence of bases which all meet a certain optimality condition, but whose induced solution $B^{-1}b$ may not be feasible for (1). In the course of the iteration, bases which improve feasibility while maintaining optimality are sought until a basis is found, which is both optimal and feasible. An outline focusing solely on the steps relevant for our context is as follows:

Algorithm 3.3 *Dual simplex algorithm.* The following steps are performed until some termination criterion is fulfilled:

[4] We will omit the word "revised" throughout. It should be clear that we are not considering a tableaux-based simplex method.

Pricing:	Based on the solution of the previous FTRAN, select a leaving index $p \in \mathcal{B}$. Depending on the pricing strategy, it may be necessary to solve a linear system with B.
BTRAN:	Solve $B^T h = (I_m)_{\bullet p}$.
Ratio test:	Based on the previous solution, select an entering index $q \in \mathcal{N}$.
FTRAN:	Solve $Bf = (A, I_m)_{\bullet q}$.
Update:	$\mathcal{B} = (\mathcal{B} \setminus \{p\}) \cup \{q\}, \quad \mathcal{N} = (\mathcal{N} \setminus \{q\}) \cup \{p\}$.

In each step of this algorithm, two indices $p \in \mathcal{B}$ and $q \in \mathcal{N}$ are exchanged in order to improve the solution that is defined by the current basis. In terms of the basis matrix B, this means that one column of B is replaced by one column of the matrix $(A, I_m)_{\bullet \mathcal{N}}$, and this exchange requires solving (at least) two linear systems, one with B^T (BTRAN) and one with B (FTRAN). Note that these solves cannot be performed simultaneously, since the FTRAN depends on the ratio test, which in turn depends on the outcome of the BTRAN.

Often, the dual simplex algorithm is started with the so-called slack basis, which means that initially $\mathcal{B} = \{n+1, \ldots, m+n\}$ and so $B = I_m$. Then in every iteration one column of the basis matrix is exchanged with one non-basic column of the matrix (A, I_m).

Since the constraint matrix A usually is very sparse, the basis matrix B will remain very sparse throughout the execution of the simplex algorithm. We remark that the right hand sides of all linear systems to be solved during the run of the simplex algorithm are very sparse as well, since each of these is either a column of the sparse matrix A, or a column of I_m (the exploitation of the sparsity of the right hand side is very important for an efficient implementation; see [Hall and McKinnon 2005; Wunderling 1996]). As a typical example, see Figure 1, which shows the nonzero pattern of the basis matrix B in step 4,835 of our run of SoPlex applied to the LP relaxation of the problem momentum1 from MIPLIB. The matrix is of order 11,633 and has 43,451 nonzero entries, meaning that it has approximately 3.7 nonzeros in each column.

The most widely employed algorithmic kernel for the solution of the systems in BTRAN and FTRAN is an LU decomposition of B. Since B is sparse, one seeks triangular matrices L and U as sparse as possible, and permutation matrices P and Q such that $LU = PBQ^T$. The problem of computing factors with a minimal number of nonzero elements is NP-complete [Yannakakis 1981]. Consequently, numerous heuristics have been developed and many different codes for computing sparse LU factorizations exist; see the exhaustive list from the University of Florida[5].

Within the simplex algorithm, the factors L and U are typically used in combination with an updating scheme that allows re-use over several iterations. Examples are given by the Forrest and Tomlin [1972], whose procedure focusses on maintaining sparsity in the factors, and the procedure by Bartels and Golub [1970], which emphasises numerical stability. Most commonly, a variant of the Forrest-Tomlin update is employed [Suhl and Suhl 1993]. Another popular updating scheme de-

[5] `http://www.cise.ufl.edu/research/sparse/codes/`

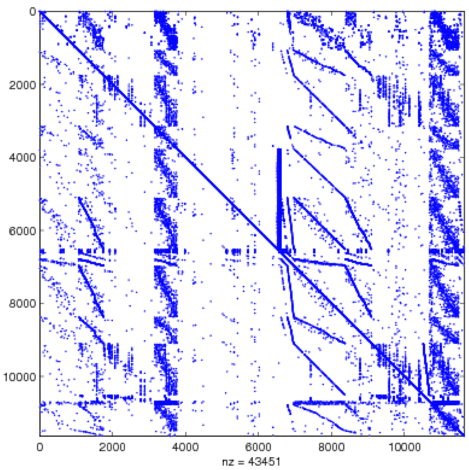

Fig. 1. Nonzero pattern of a typical basis matrix B in the simplex algorithm.

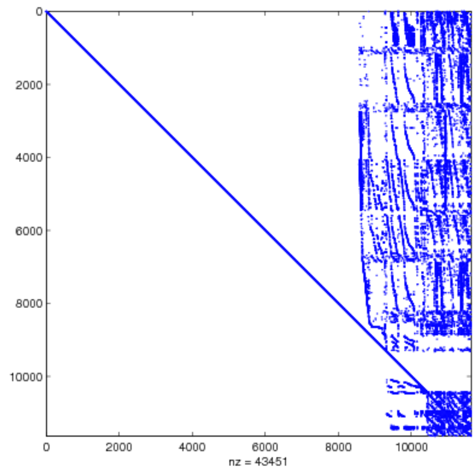

Fig. 2. Nonzero pattern of the matrix in Figure 1 after permuting singletons to the front.

rives from the "product form of the basis inverse" (PFI). A recent presentation of the PFI update under consideration of computational aspects can be found in [Maros 2003].

3.2 Structure of basis matrices

At first sight, the nonzero pattern of LP basis matrices seems to lack any structure (cf. Figure 1). In particular, it was observed that LP basis matrices are particularly nonsymmetric, see [Duff et al. 1986, p. 123]. This is not surprising, since B can be regarded as a random selection (by the pricing step) of unit vectors and columns of A, which itself usually has no special structure besides maybe some visually identifiable pattern stemming from the model the LP represents. Consequently, during the run of the simplex algorithm, the nonzero pattern of B is completely unpredictable.

However, as indicated in the preceding section, a well known distinguishing property of LP basis matrices in comparison with matrices arising in other application areas is that often a large part can be triangulated by means of permutations (again, see [Orchard-Hays 1968; Tomlin 1972]). This is trivially true when initially $B = I_m$, and it remains true for many simplex steps, often throughout until termination, as we will demonstrate numerically for a wide range of different LPs in section 4.2. Mathematically, this means that by successively moving column- or row singletons to the front, we obtain permutation matrices P, Q such that the permuted basis matrix PBQ is of the form

$$\begin{pmatrix} U^0 & * & * \\ 0 & L^0 & 0 \\ 0 & * & N \end{pmatrix}, \tag{2}$$

with a lower triangular matrix L^0, an upper triangular matrix U^0 and a matrix N, called the *nucleus* (or *kernel*), which contains no column- or row singleton.

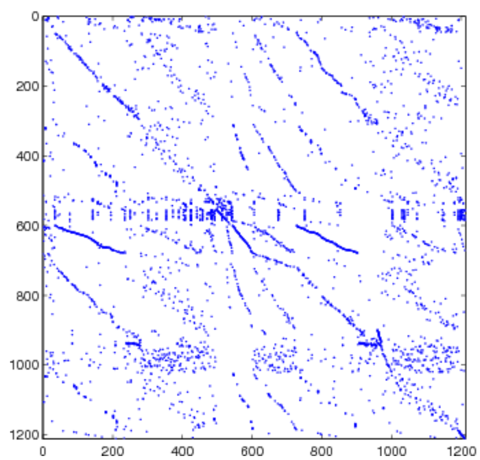

Fig. 3. Nonzero pattern of the nucleus N of the matrix in Figure 2.

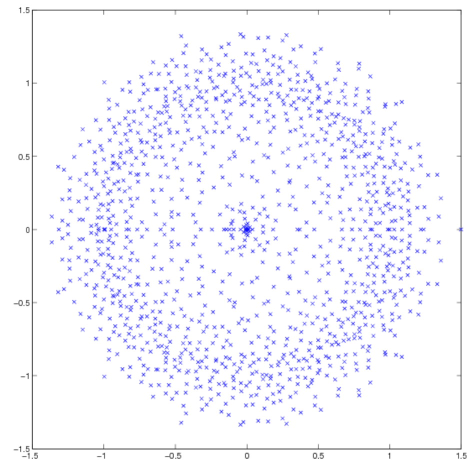

Fig. 4. Spectrum of the nucleus N from Figure 3.

Of course, the matrix U^0 comprises all unit columns of B which may, or may not, account for a large part of U^0. Thus with appropriate permutations, LP basis matrices exhibit a very pronounced structure which is crucial for efficient solution of the linear systems. As an example, see Figure 2, which shows the nonzero structure of the permuted basis matrix B from Figure 1. The matrix contains 3,063 non-unit columns, the nucleus N is of order 1,211, which is about 10% of the order of B, and has 3,923 nonzero entries.

Figure 3 shows the nonzero pattern of N. This matrix cannot be triangulated by means of permutations, and we are unable to determine any special feature, except for the fact that N is sparse. In general, the nuclei appear to be sparse, nonsymmetric, and indefinite (with eigenvalues on both sides of the imaginary axis). As an example, see Figure 4, where we show the spectrum of N from Figure 3.

3.3 Remark on iterative solvers

Since L^0 and U^0 are both triangular, the only non-trivial part for solving the systems in the FTRAN and BTRAN are solves with the nucleus N. We will show in section 4.2 below that the dimension of N typically is significantly smaller than the dimension of B. When the dimension of N is too small (less than 10^4, say), it is unlikely that any iterative solver will outperform a direct solver. Moreover, the favorable property of modern iterative solvers that they can operate matrix-free, i.e., without requiring the matrix to be stored in memory, does not apply in the LP context, where already the constraint matrix A is explicitly stored. Finally, the spectra of the nuclei N we looked at indicate rather unfavorable convergence behavior of iterative methods (cf. Figure 4), unless a very good preconditioner is used. But even in the case where such preconditioner is obtained "for free", our results in section 4.3, which show that the fill-in in the LU factorization of N obtained by dynamic Markowitz pivoting is very low, imply that in order to be competitive with a direct solver, a preconditioned iterative solver would have to compute a good approximate solution within very few iterations.

4. NUMERICAL EXPERIMENTS

4.1 The set of LPs used in our experiments

For our numerical experiments, we used LPs from four different sources:

(1) The NETLIB set of real-world LPs (94 LPs). Although this publicly available test set dates back many years and most of the problems are solved within a fraction of a second, we consider the ones of larger size to be interesting for our purpose of comparing fill-in in the factors.
(2) The MIPLIB 2003 test set of mixed-integer linear programs (60 LPs). In order to mimic a typical root relaxation for branch-and-bound based MIP algorithms, we solve the resulting LP after applying CPLEX[6] MIP-presolve and relaxing integrality constraints.
(3) The LPs from the Mittelmann benchmark of free LP solvers that do not come from source 1 or 2 (35 LPs).
(4) Large-scale LPs, mostly with the LP basis dimension exceeding $5 \cdot 10^5$, provided to us by ZIB (11 LPs).

From these sets, we selected all instances of basis dimension (i.e., the number of rows) greater than or equal to 10^3 for our numerical experiments. Of course, this number can be regarded to be chosen quite arbitrarily, but it provides means to filter out LPs that are nowadays of limited computational relevance while a broad range of real world LPs is maintained in the test set.

Table I gives an overview of the 88 LPs. An LP name containing a "*" indicates that the full name has been shortened to save table space. In addition to the number of rows, columns and non-zeros, it shows the density of the LP constraint matrix, the average number of non-zeros per column (column "∅nzpc") and in which set the LP can be found (numbers refer to the list above). It should be noted that all numbers refer to the LP after SoPlex has performed its presolve procedure, so that the number of rows shown matches the basis dimension for the computational steps in the simplex algorithm.

name	nrows	ncols	nnz	%dens	∅nzpc	source
80bau3b	1990	8778	19157	0.110	2.18	1
a1c1s1	2283	2619	8156	0.136	3.11	2
aflow40b	1405	2691	6709	0.177	2.49	2
aflo*0_50	500998	998000	2494495	0.000	2.50	4
aflo*0_50	2001998	3996000	9988972	0.000	2.50	4
atla*a-ip	19446	17343	179287	0.053	10.34	2
BER_*od10	1425456	558174	4941366	0.001	8.85	4
bnl2	1559	2702	11177	0.265	4.14	1
cap6000	1891	4689	14044	0.158	3.00	2
classify	21398	22805	5852594	1.199	256.64	4

continued on next page

[6]http://www.ilog.com/products/cplex

continued from previous page

name	nrows	ncols	nnz	%dens	nzpc	source
cont1	120395	40398	359593	0.007	8.90	3
cont11	120395	80396	359593	0.004	4.47	3
cont11_l	1468599	981396	4403001	0.000	4.49	3
cont1_l	1918399	641598	5752001	0.000	8.97	3
cont4	106866	39602	331739	0.008	8.38	3
cycle	1277	2132	13969	0.513	6.55	1
d2q06c	2021	4759	30947	0.322	6.50	1
dano3mip	3151	13837	79001	0.181	5.71	2
dbic1	33688	140320	781909	0.017	5.57	3
degen3	1502	1817	24644	0.903	13.56	1
dfl001	5927	11165	34085	0.052	3.05	1
fit2p	3000	10525	47284	0.150	4.49	1
fome12	23708	44660	136340	0.013	3.05	3
fome13	47416	89320	272680	0.006	3.05	3
ganges	1097	1360	6290	0.422	4.62	1
gen4	1537	4297	107102	1.622	24.92	3
gesa2	1344	1176	4968	0.314	4.22	2
gesa2-o	1176	1152	3648	0.269	3.17	2
greenbea	1858	3879	23418	0.325	6.04	1
greenbeb	1854	3864	23350	0.326	6.04	1
ken-18	78862	128304	298728	0.003	2.33	3
l30	2701	15380	51169	0.123	3.33	3
liu	2178	1154	10626	0.423	9.21	2
lp22	2872	8693	60181	0.241	6.92	3
manna81	6480	3321	12960	0.060	3.90	2
maros-r7	2156	6620	80480	0.564	12.16	1
mkc	1286	3223	12509	0.302	3.88	2
mod011	1404	7022	13969	0.142	1.99	2
mod2	29882	29316	136227	0.016	4.65	3
momentum1	11633	3579	46429	0.112	12.97	2
momentum2	18840	3306	178920	0.287	54.12	2
momentum3	53868	13333	542736	0.076	40.71	2
msc98-ip	15008	12797	79499	0.041	6.21	2
mzzv11	8272	8775	114289	0.157	13.02	2
mzzv42z	9951	11291	136659	0.122	12.10	2
N_BA*mann	3160202	1573827	7869027	0.000	5.00	4
neos	423189	36786	915386	0.006	24.88	3
neos1	131581	1892	468009	0.188	247.36	3
neos2	131902	1560	549855	0.267	352.47	3
neos3	512209	6624	1542816	0.045	232.91	3
net12	13757	13819	78232	0.041	5.66	2
nsct2	7797	11297	612106	0.695	54.18	3
nug08-3rd	19728	20448	139008	0.034	6.80	3
nug15	6330	22275	94950	0.067	4.26	3

continued on next page

continued from previous page

name	nrows	ncols	nnz	%dens	nzpc	source
nug20	15240	72600	304800	0.028	4.20	3
pds-100	152300	489909	1053001	0.001	2.15	3
pds-40	64276	210139	454345	0.003	2.16	3
pilot	1373	3337	40653	0.887	12.18	1
pilot87	1967	4587	70372	0.780	15.34	1
protfold	2110	1835	21776	0.562	11.87	2
qiu	1192	840	3432	0.343	4.09	2
rail4284	4176	1090538	11174651	0.245	10.25	3
rd-r*c-21	54169	538	352274	1.209	654.78	2
rlfprim	57422	8048	264483	0.057	32.86	3
roll3000	1109	810	20432	2.275	25.22	2
scm3*0pre	1220936	3602518	14407840	0.000	4.00	4
sctap3	1344	1767	7630	0.321	4.32	1
seymour	4624	1085	32282	0.643	29.75	2
sgpf5y6	143546	206033	500901	0.002	2.43	3
sierra	1212	2016	7242	0.296	3.59	1
sp97ar	1670	14085	276989	1.178	19.67	2
spal_004	10203	321696	46161316	1.406	143.49	3
stat96v1	5846	190755	581635	0.052	3.05	3
stat96v2	28750	942131	2835917	0.010	3.01	3
stat96v4	3172	62211	490471	0.249	7.88	3
stocfor2	2065	1951	8127	0.202	4.17	1
stocfor3	16105	15151	63543	0.026	4.19	1
stor*-125	56286	138619	376373	0.005	2.72	3
stor*1000	450036	1108119	3008373	0.001	2.71	3
stp3d	97476	136940	498355	0.004	3.64	2
truss	1000	8806	27836	0.316	3.16	1
ts.l*0315	1654588	271971	3764552	0.001	13.84	4
ts.l*2029	1089131	216935	2533576	0.001	11.68	4
ts.l*2253	1089128	216875	2533513	0.001	11.68	4
ts.l*4012	1654588	271980	3764561	0.001	13.84	4
ts.l*4139	2214771	310996	4991762	0.001	16.05	4
watson_2	206926	400438	1083142	0.001	2.70	3
world	29768	31061	137057	0.015	4.41	3

Table I: Information on the LPs for the numerical experiments

4.2 The size of the nucleus N

We already indicated above that the dimension of the nucleus N typically is significantly smaller than the dimension of the corresponding basis matrix B. Our experimental setup is as follows: we let SoPlex solve each of the LPs (with a time limit of $80,000$ seconds). Whenever SoPlex decided to compute a factorization of the basis matrix, we recorded the number of non-unit columns in the basis and the dimension of the nucleus at that iteration. In order to obtain a single number for both quantities, we simply use the average over all such factorizations where the nucleus did not vanish, that is, where the basis matrix was not a (permuted) triangular matrix. All this information is shown in Table II: The second column shows how many factorizations were computed and the third how many of these resulted in a non-vanishing nucleus. Next to the basis dimension of the LP (which is, of course, constant throughout the execution of the simplex algorithm) columns five and six show the average number of non-unit columns and average nucleus dimension. Again, the average is taken over all factorizations where the nucleus did not vanish to allow for comparability among columns five and six.

LP name	#fac	$\#N$	dim B	∅ #NU	∅ dim N
80bau3b	28	25	1990	1271	139
a1c1s1	8	2	2283	1085	43
aflow40b	14	7	1405	1340	18
aflo*0_50	2658	210	500998	490697	584
aflo*0_50	7342	11	2001998	1461779	31
atla*a-ip	85	84	19446	4891	1398
BER_*od10	834	833	1425456	20107	9602
bnl2	12	11	1559	750	246
cap6000	6	5	1891	295	2
classify	183	182	21398	335	335
cont1	212	113	120395	30913	22329
cont11	577	478	120395	60996	35746
cont11_l	1660	438	1468599	288100	86164
cont1_l	2064	467	1918399	366000	93375
cont4	208	110	106866	30292	21212
cycle	6	5	1277	370	124
d2q06c	33	32	2021	1268	753
dano3mip	180	179	3151	1822	1013
dbic1	63417	63416	33688	7991	932
degen3	19	17	1502	935	506
dfl001	115	112	5927	4595	2254
fit2p	30	29	3000	21	20
fome12	440	437	23708	18140	8886
fome13	901	895	47416	36295	17793
ganges	8	7	1097	628	190
gen4	8	7	1537	526	526
gesa2	8	7	1344	410	99
gesa2-o	7	6	1176	365	70
greenbea	63	62	1858	1558	630
greenbeb	36	35	1854	1399	603
ken-18	885	318	78862	66953	78
l30	69	68	2701	2347	2340
liu	4	1	2178	408	2
lp22	146	145	2872	1707	1518
manna81	17	16	6480	1522	295

continued on next page

continued from previous page

LP name	#fac	$\#N$	dim B	∅ #NU	∅ dim N
maros-r7	15	14	2156	924	907
mkc	6	4	1286	464	31
mod011	13	1	1404	716	7
mod2	355	353	29882	11573	2884
momentum1	44	38	11633	2687	760
momentum2	52	48	18840	2243	624
momentum3	149	116	53868	8934	4444
msc98-ip	79	78	15008	3882	1207
mzzv11	666	665	8272	3355	2384
mzzv42z	111	110	9951	3893	1200
N_BA*mann	4879	73	3160202	28152	34
neos	464	463	423189	27682	2213
neos1	46	44	131581	1393	453
neos2	70	68	131902	1363	597
neos3	435	431	512209	5908	4667
net12	23	22	13757	1025	433
nsct2	34	33	7797	251	162
nug08-3rd	7946	7945	19728	10214	9491
nug15	43227	43226	6330	5671	5544
nug20	12308	12307	15240	9256	8181
pds-100	2365	2113	152300	103632	3907
pds-40	541	294	64276	56506	1607
pilot	28	27	1373	1030	863
pilot87	66	65	1967	1496	1317
protfold	13	12	2110	466	308
qiu	7	6	1192	562	204
rail4284	305	304	4176	2456	2179
rd-r*c-21	3	2	54169	261	126
rlfprim	48	45	57422	3917	372
roll3000	6	5	1109	264	145
scm3*0pre	2441	2438	1220936	45501	22692
sctap3	4	3	1344	212	13
seymour	13	12	4624	497	345
sgpf5y6	685	634	143546	72758	290
sierra	5	0	1212	0	0
sp97ar	8	7	1670	245	215
spal_004	1731	1730	10203	1564	1564
stat96v1	88	87	5846	4811	4694
stat96v2	741	740	28750	25612	25228
stat96v4	515	514	3172	2776	2748
stocfor2	11	10	2065	845	70
stocfor3	78	75	16105	6334	584
stor*-125	517	464	56286	35596	1509
stor*1000	4233	4110	450036	274215	12286

continued on next page

continued from previous page

LP name	#fac	$\#N$	$\dim B$	∅ #NU	∅ $\dim N$
stp3d	6481	6357	97476	53783	17763
truss	99	98	1000	945	704
ts.l*0315	913	911	1654588	78578	3684
ts.l*2029	664	663	1089131	57115	1846
ts.l*2253	739	738	1089128	63732	3186
ts.l*4012	1368	1366	1654588	103807	12762
ts.l*4139	1006	1005	2214771	87748	3713
watson_2	2055	2053	206926	146320	21970
world	478	477	29768	12495	3230

Table II: Detailed data on the nuclei sizes

Observe that for some LPs (the "aflow" LPs, for example), the basis matrix often really is a permuted triangular matrix. In this case, there is no nucleus and the matrix admits a trivial factorization. From the difference between column 2 and column 3 it can be seen how often this happens for a particular LP instance.

Fig. 5 offers a more condensed view of the data in Table II: In the upper plot, our 88 LPs are given on the x-axis, sorted by their average nuclei size as in column 6 of Table II. The y-axis shows the average nucleus size in percentage of the basis dimension,

$$100 * \frac{\varnothing \dim N}{\dim B},$$

so every cross ($\times$) in this plot maps an LP to this quantity. Associated with every such cross, a vertical interval is shown, which simply depicts the mean deviation of the relative nuclei sizes from their average. Figure 6 shows the same data as Figure 5, but restricted to the 35 LPs having the smallest average nucleus sizes. We see that for a considerable number of LPs, including all large-scale LPs, the nuclei sizes never reach 4% of the size of the basis matrix throughout the simplex run. Note that in contrast to the numbers on the average nucleus dimension in Table II, the averages shown in these figures also take into account the occurrences of vanishing nuclei.

4.3 Fill-in results for dynamic Markowitz pivoting (SoPlex)

The LU factorization of SoPlex is based on a right-looking scheme using dynamic Markowitz pivoting, similar to what is described in [Suhl and Suhl 1990]. Pivoting (for sparsity and stability) is performed on the whole remaining submatrix at every stage using current row and column counts. No column ordering is symbolically computed upfront and no BLAS is used.

We applied SoPlex to each of the LPs in our test set. We recorded the number of non-zeros in the L and U factors of the nucleus N of every basis matrix B at the iteration steps of the simplex algorithm where B was factorized. Let the number of nonzero elements in the factors of N computed by SoPlex be denoted by

$$n_s := \mathrm{nnz}(L - I) + \mathrm{nnz}(U).$$

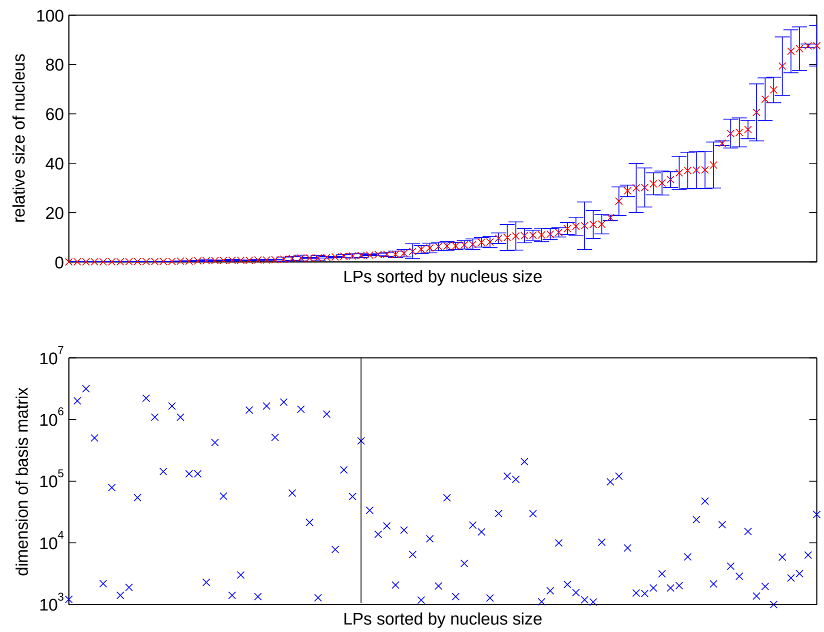

Fig. 5. Upper plot: Ranges of the nuclei sizes of the LPs we tested. The LPs are sorted according to nucleus size. The lower graph indicates the dimension of the corresponding LP.

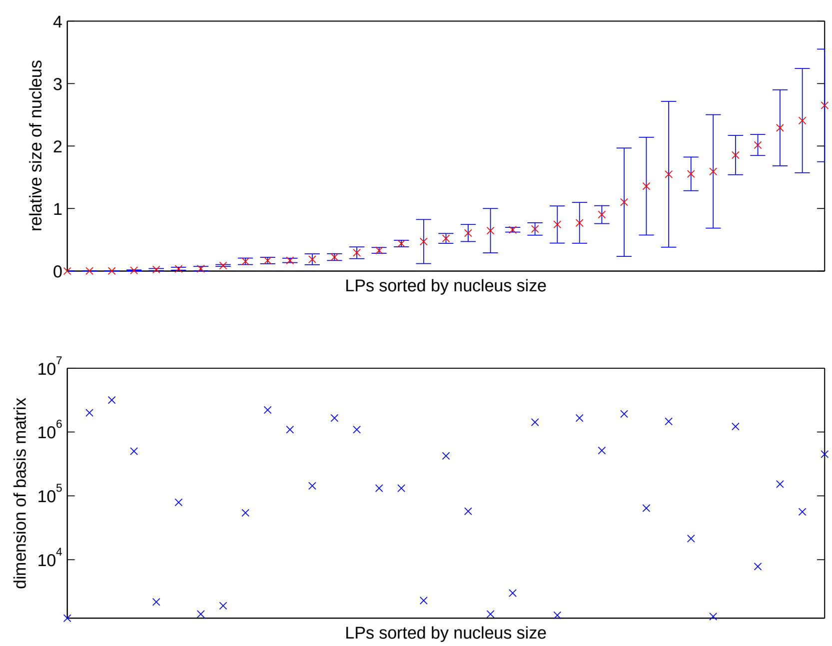

Fig. 6. The LPs from Figure 5 on the left-hand side of the marker.

Then $\frac{n_s}{\mathrm{nnz}(N)}$ is the relative fill-in in the factorization of N. Furthermore, let the number of nonzero elements in the triangular parts of B permuted as in (2), be denoted by

$$n_t := \mathrm{nnz}(B) - \mathrm{nnz}(N).$$

Then $\frac{n_s+n_t}{\mathrm{nnz}(B)}$ is the relative fill-in in the factorization of B. In order to obtain a single number for each LP, we computed the average of these numbers over all factorized basis matrices and nuclei, respectively.

Table III shows these two different fill measures in columns three and six alongside with the mean deviation of the relative fill-in from the average. For completeness, the number of factorizations of B and N are shown in columns two and five. Note that the average fill-in for the LP instances "ganges" and "nsct2" is less than 1.0. The reason is numerical cancellation in the course of the factorization.

LP name	#fac. B	∅fill B	dev.	#fac. N	∅fill N	dev.
80bau3b	28	1.016	0.008	25	1.230	0.029
a1c1s1	8	1.001	0.002	2	1.301	0.009
aflow40b	14	1.002	0.002	7	1.264	0.037
aflo*0_50	2658	1.000	0.000	210	1.425	0.134
aflo*0_50	7342	1.000	0.000	11	1.436	0.019
atla*a-ip	85	1.026	0.020	84	1.500	0.241
BER_*od10	834	1.015	0.004	833	1.774	0.114
bnl2	12	1.051	0.022	11	1.291	0.043
cap6000	6	1.000	0.000	5	1.000	0.000
classify	183	1.049	0.017	182	3.864	0.173
cont1	212	2.348	1.566	113	7.424	2.699
cont11	577	5.292	2.234	478	11.452	2.211
cont11_l	1660	1.219	0.340	438	6.120	1.978
cont1_l	2064	1.116	0.187	467	5.907	1.875
cont4	208	2.267	1.491	110	7.072	2.679
cycle	6	1.022	0.010	5	1.244	0.046
d2q06c	33	1.189	0.105	32	1.495	0.177
dano3mip	180	1.506	0.260	179	2.551	0.482
dbic1	63417	1.013	0.002	63416	1.397	0.031
degen3	19	1.022	0.011	17	1.139	0.043
dfl001	115	1.342	0.175	112	1.795	0.256
fit2p	30	1.000	0.000	29	1.003	0.059
fome12	440	1.357	0.193	437	1.813	0.295
fome13	901	1.350	0.184	895	1.798	0.278
ganges	8	0.999	0.003	7	1.008	0.041
gen4	8	3.979	2.291	7	8.335	2.783
gesa2	8	1.005	0.003	7	1.071	0.028
gesa2-o	7	1.001	0.001	6	1.015	0.017
greenbea	63	1.132	0.031	62	1.456	0.049
greenbeb	36	1.147	0.041	35	1.477	0.056
ken-18	885	1.000	0.000	318	1.383	0.075

continued on next page

continued from previous page

LP name	#fac. B	∅fill B	dev.	#fac. N	∅fill N	dev.
l30	69	4.376	0.960	68	4.872	1.001
liu	4	1.000	0.000	1	1.000	0.000
lp22	146	3.351	0.955	145	4.715	1.156
manna81	17	1.006	0.009	16	1.167	0.000
maros-r7	15	1.501	0.134	14	1.884	0.096
mkc	6	1.002	0.003	4	1.062	0.062
mod011	13	1.000	0.000	1	1.111	0.000
mod2	355	1.050	0.020	353	1.508	0.061
momentum1	44	1.022	0.016	38	1.409	0.090
momentum2	52	1.006	0.004	48	1.370	0.087
momentum3	149	1.022	0.012	116	1.625	0.099
msc98-ip	79	1.033	0.019	78	1.420	0.133
mzzv11	666	1.190	0.055	665	1.895	0.163
mzzv42z	111	1.045	0.015	110	1.428	0.075
N_BA*mann	4879	1.000	0.000	73	1.243	0.068
neos	464	1.001	0.001	463	1.305	0.092
neos1	46	1.003	0.002	44	1.899	0.308
neos2	70	1.004	0.003	68	2.035	0.538
neos3	435	1.020	0.016	431	2.947	1.340
net12	23	1.015	0.006	22	1.380	0.049
nsct2	34	0.995	0.001	33	0.839	0.016
nug08-3rd	7946	13.419	1.470	7945	27.648	2.381
nug15	43227	15.311	0.642	43226	17.588	0.597
nug20	12308	15.733	3.604	12307	24.370	3.397
pds-100	2365	1.011	0.006	2113	1.414	0.038
pds-40	541	1.006	0.007	294	1.399	0.046
pilot	28	2.083	0.474	27	2.453	0.367
pilot87	66	2.546	0.420	65	2.863	0.303
protfold	13	1.150	0.074	12	1.872	0.248
qiu	7	1.052	0.071	6	1.204	0.135
rail4284	305	2.069	0.294	304	2.972	0.436
rd-r*c-21	3	1.000	0.000	2	1.242	0.031
rlfprim	48	1.003	0.002	45	1.397	0.060
roll3000	6	1.013	0.008	5	1.123	0.030
scm3*0pre	2441	1.031	0.014	2438	1.674	0.109
sctap3	4	1.000	0.000	3	1.017	0.011
seymour	13	1.021	0.012	12	1.424	0.115
sgpf5y6	685	1.000	0.000	634	1.235	0.033
sierra	5	1.000	0.000	0	0.00	0.00
sp97ar	8	1.041	0.023	7	1.145	0.041
spal_004	1731	6.596	3.697	1730	27.699	12.051
stat96v1	88	2.220	0.491	87	2.363	0.375
stat96v2	741	4.520	1.095	740	4.697	0.946
stat96v4	515	1.716	0.219	514	1.751	0.185

continued on next page

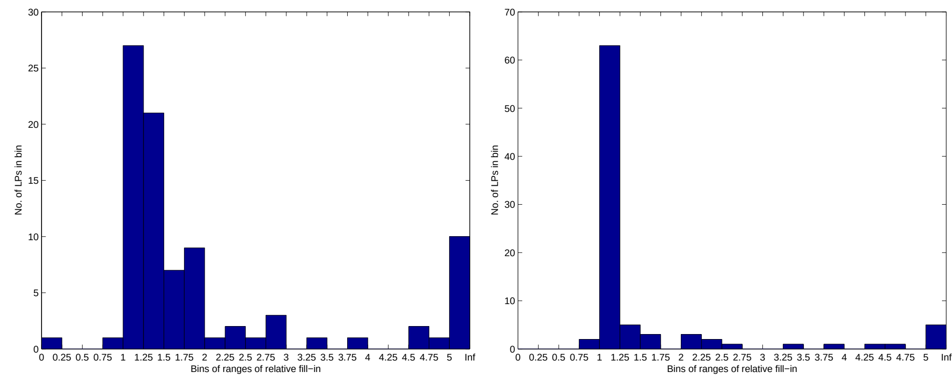

Fig. 7. Distribution of the LPs by their average fill-in in the nucleus (left) and basis matrix (right) at iterations where the nucleus was factorized. See section 4.3 for explanations.

continued from previous page

LP name	#fac. B	∅fill B	dev.	#fac. N	∅fill N	dev.
stocfor2	11	1.009	0.009	10	1.213	0.120
stocfor3	78	1.010	0.008	75	1.215	0.092
stor*-125	517	1.007	0.005	464	1.231	0.059
stor*1000	4233	1.008	0.006	4110	1.249	0.061
stp3d	6481	1.413	0.119	6357	3.264	0.522
truss	99	1.379	0.074	98	1.515	0.087
ts.l*0315	913	1.001	0.001	911	1.160	0.040
ts.l*2029	664	1.001	0.000	663	1.131	0.032
ts.l*2253	739	1.001	0.001	738	1.170	0.051
ts.l*4012	1368	1.005	0.005	1366	1.257	0.117
ts.l*4139	1006	1.001	0.001	1005	1.175	0.059
watson_2	2055	1.069	0.039	2053	1.515	0.111
world	478	1.058	0.021	477	1.538	0.084

Table III: #fac. indicates the number of SoPlex factorizations of the basis matrices B and N, "fill" shows the average fill-in during the LP process and "dev." indicates the mean deviation.

A more condensed view of the data in Table III is offered in Figures 7. These histograms show the distribution of the average relative fill-in in B and N over the whole set of 88 LPs. That means that each LP accounts for one data item and is sorted into the bin according to its average fill-in. Note that for the LP "sierra" there was not a single nucleus to factorize, so that this LP accounts for the leftmost bin in Figure 7 by convention. The LP's which account for the elements in the bins covering the interval $[0.75, 1.0)$ in both Figures are "ganges" and "nsct2": As explained before, numerical cancellation during the factorization results in factors L and U that are slightly sparser than the nuclei N or basis matrices themselves.

From Figure 7 it is immediately clear that the number of fill-in elements in N

generated by SoPlex' LU factorization of N is extremely small: for about 75% of the LPs, the average relative fill-in is less than two, and for many of them much less. Combined with the results on the nucleus sizes from section 4.2, the distribution shown in right graphic in Figure 7 is no surprise as the average relative fill-in in the basis matrices is close to 1.0 for almost all LPs in our set.

Of the ten LPs of which the average fill-in in the nucleus is quite large, say, greater than five as indicated by the rightmost bin in Figure 7, the "nug*" LPs account for three and the "cont*" LPs account for four. Since one can expect that different instances of the same underlying model share many structural properties, it may be advisable to employ a counting scheme that covers this aspect so as not to "overweight" similar LPs. In order not to make the presentation of the data overly complicated, we did not use such a counting scheme, but expect the reader to consider this aspect when judging based on the data shown in Figure 7.

We remark that within the runs of SoPlex for the 88 test LPs a total of 181,460 basis matrices B were factorized, and 162,174 had a nontrivial nucleus N.

4.4 Comparison with other LU codes

We now describe the fill-in results for the factorization of the nucleus N generated by a selection of modern LU codes and compare them with the results produced by SoPlex. In our selection of LU codes, we tried to achieve a good coverage of top-level strategies different from what is employed in SoPlex. We used the exhaustive list of available LU codes maintained at the University of Florida[7] for orientation, which led us to use Pardiso 3.1[8], Umfpack 5.0.1[9], and Wsmp 6.9.25[10]. We did not select a LU code based on (full) dynamic Markowitz pivoting, because we expect the resulting number of fill elements would be very similar to the number of fill elements generated by SoPlex' built-in factorization routines.

When a nonsymmetric, sparse linear system of equations $Ax = b$ is to be solved by means of an LU factorization, it is a standard procedure to permute the matrix A into block triangular form (BTF) [Duff et al. 1986, ch. 6] first, so that only the diagonal blocks need to be factorized. This preprocessing can greatly reduce the fill-in if the matrix is far from being irreducible, that is, if the graph of the matrix A is far from being strongly connected.

In the very special setting of this LP context, BTF is not applicable: the column exchange in B from one iteration to the next can alter the strongly connected components of the graph of the basis matrix quite drastically, so that efficient updating procedures for the LU factors of the diagonal blocks of the BTF would be very difficult to develop.

Wsmp and Pardiso both compute their default fill-in reducing ordering based on the pattern of the symmetric matrix $N^T + N$. The primary advantage of this strategy is that the numerical factorization can be computed very efficiently. Due to the lack of structural symmetry in N, these methods produce more fill-in than methods based on Markowitz pivoting. The fact that the factorization of LP basis

[7] `http://www.cise.ufl.edu/research/sparse/codes/`
[8] `http://www.pardiso-project.org`
[9] `http://www.cise.ufl.edu/research/sparse/umfpack/`
[10] `http://www-users.cs.umn.edu/~agupta/wsmp.html`

matrices cannot be cast as the factorization of the diagonal blocks of their block triangular form poses a difficulty for WSMP. UMFPACK pre-orders the columns of N and performs partial Markowitz pivoting in the course of the numerical factorization. This explains why UMFPACK delivers factors with only slightly more fill-in than (full) dynamic Markowitz pivoting.

4.4.1 *Methodology.* For our comparison we let SoPlex solve each LP as usual (with a time limit of $80,000$ seconds for each LP). Every time SoPlex factorizes a basis matrix, we let the other LU packages factorize it as well. For every factorization we recorded the fill-in. Apart from counting the fill-in, the factorizations of the other LU codes were not used at all. All algorithmic decisions remained to be determined by the solves with SoPlex' own factorization.

We benchmarked one LU code at a time, that is, only one LU code was called from SoPlex during the solution of the LPs. This procedure was repeated for each of the LU codes. All computations were performed on a 64 bit Linux box with two AMD Opteron 252 CPU with a clock rate of 2.6 GHz and 4GB memory. Only one CPU was used for each run and all LPs were solved sequentially. Some LU codes offer a multithreaded implementation, but we always used the codes in serial mode. The parameters of the LU codes were mostly set to the default values. Packages that use a threshold for pivoting were instructed to use the value 0.01, which is the default value SoPlex uses. WSMP was set up not to perform the reduction to BTF, for the reasons explained above.

4.4.2 *Results.* Consider the basis matrix B at a SoPlex iteration where it is factorized. We define $n_t := \mathrm{nnz}(B) - \mathrm{nnz}(N)$ as in section 4.3. Analogously to n_s as in section 4.3, let n_X for LU code X denote the number $n_X := \mathrm{nnz}(L-I) + \mathrm{nnz}(U)$, where now code X (instead of SoPlex' LU factorization) is used to compute the L and U factors of N. Since the computational complexity of solving a system with resulting factorization of B is a linear function of $n_X + n_t$, the number

$$\frac{n_X + n_t}{n_s + n_t}$$

measures the relative change in the complexity for the solve if the LU factorization from code X was used as a drop-in replacement for SoPlex' own factorization. For example, a value of 1.5 means that the solves with B would take 50% more operations if X was used.

Another number of interest, which measures the change in the relative fill-in, is n_X/n_s. For example, a value of 1.5 means that code X has produced 50% more fill-in elements per nonzero in N than SoPlex.

To obtain a single number over a whole SoPlex run, we take the geometric mean over all such basis matrices for both quantities. We denote the geometric mean over all $(n_X + n_t)/(n_s + n_t)$ by c_B, and the geometric mean over all (n_X/n_s) by c_N.

Figures 8–11 show histograms of the distributions of c_N and c_B for the three LU codes used in this benchmark (in all plots, the rightmost bin accounts for values greater than five). From the c_N distributions we see that PARDISO and WSMP generated in average twice as much fill-in elements as SoPlex which can be explained by the symmetric minimum degree algorithm used by PARDISO instead of the dynamic Markowitz pivoting. UMFPACK generated more fill-in for far fewer LPs than the other LU packages and even performed a little better than SoPlex' for some LPs. But altogether UMFPACK still does slightly worse, as Figure 11 shows.

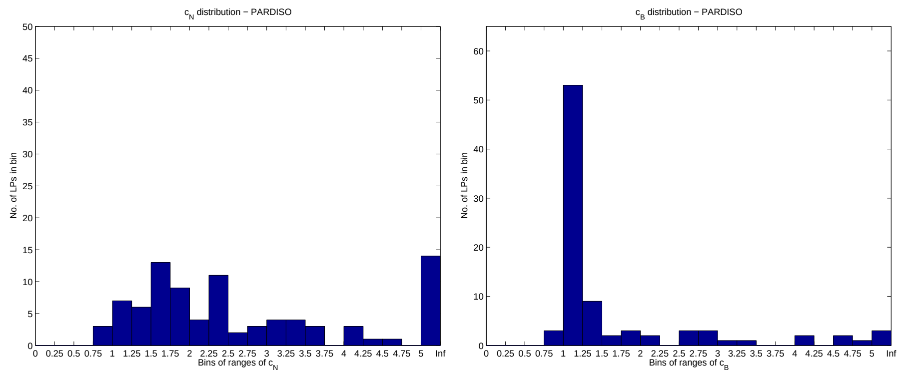

Fig. 8. Distribution of c_B and c_N for PARDISO.

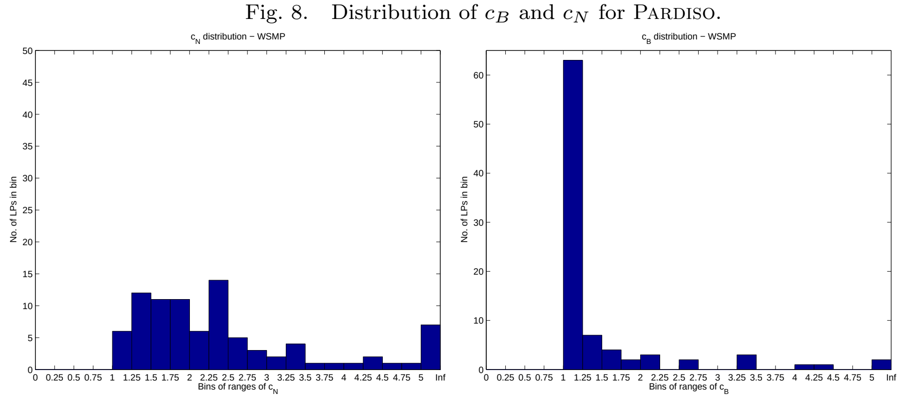

Fig. 9. Distribution of c_B and c_N for WSMP.

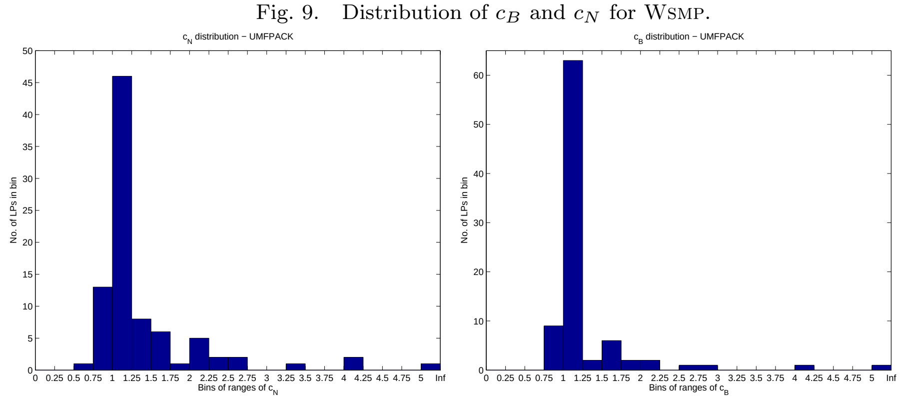

Fig. 10. Distribution of c_B and c_N for UMFPACK.

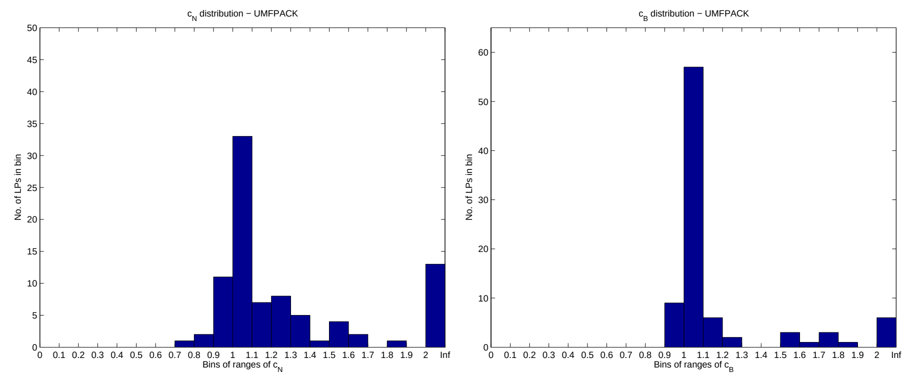

Fig. 11. Distribution of c_B and c_N for UMFPACK, on a finer scale as in Figure 10.

Note that c_B could be computed more precisely, if one would apply the LU update scheme used in SoPlex to the factors of the decomposition from the other LU code. One would obtain the quantity $(n_X + n_t)/(n_s + n_t)$ for every iteration then, and not only those at which a basis factorization takes place. But since sparser factors will tend to produce fewer additional fill-in elements during the LU update, the restriction to these iterations already delivers a good qualitative approximation, that is, whether the other LU code performs well or not compared to SoPlex.

The time limit of $80,000$ seconds was hit by some of the more difficult LPs. This implies that some caution has to be taken if one would want to compare the other LU codes against each other, since a different number of basis factorizations may have been performed. But since our intention is to compare SoPlex with each of the modern LU codes, this aspect poses no difficulty for the interpretation of our results.

Conclusions

Our numerical experiments on a wide range of LP instances, including very large-scale ones, have shown that the factorization of basis matrices can typically be considered as being very simple from the numerical point of view: The non-triangular parts of simplex basis matrices are usually very small, and an LU factorization of these parts can often be computed with only few fill-in, and thus at very low cost, using dynamic Markowitz pivoting. We believe that in this special setting it is unlikely that faster basis factorization routines for the simplex algorithm based on more sophisticated direct methods can be developed.

ACKNOWLEDGMENTS

We would like to thank Anshul Gupta for many helpful comments and advice on using WSMP.

REFERENCES

ACHTERBERG, T., KOCH, T., AND MARTIN, A. 2006. MIPLIB 2003. *Operations Research Letters 34,* 4, 1–12. See http://miplib.zib.de.

Bartels, R. H., Golub, G. H., and Saunders, M. A. 1970. Numerical techniques in mathematical programming. In *Nonlinear Programming (Proc. Sympos., Univ. of Wisconsin, Madison, Wis., 1970)*. Academic Press, New York, 123–176.

Cipra, B. A. 2000. The Best of the 20th Century: Editors Name Top 10 Algorithms. *SIAM News 33,* 4.

Davis, T. A. 2004a. Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw. 30,* 2, 196–199.

Davis, T. A. 2004b. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw. 30,* 2, 165–195.

Davis, T. A. and Hager, W. W. 2008. A sparse proximal implementation of the LP dual active set algorithm. *Math. Programming 112*, 275–301.

Duff, I. S., Erisman, A. M., and Reid, J. K. 1986. *Direct Methods for Sparse Matrices.* Oxford University Press.

Duff, I. S. and Pralet, S. 2005. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM J. Matrix Analysis and Applications 27,* 2, 313–340.

Forrest, J. J. H. and Tomlin, J. A. 1972. Updated triangular factors of the basis to maintain sparsity in the product form simplex method. *Math. Programming 2*, 263–278.

Gould, N. I. M., Scott, J. A., and Hu, Y. 2007. A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations. *ACM Trans. Math. Software 33,* 2, Art. 10, 32.

Gupta, A. 2002a. Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices. *SIAM J. of Matrix Anal. and Analysis 24,* 2, 529–552.

Gupta, A. 2002b. Recent advances in direct methods for solving unsymmetric sparse systems of linear equations. *ACM Trans. Math. Softw. 28,* 3, 301–324.

Hall, J. A. J. and McKinnon, K. I. M. 2005. Hyper-sparsity in the revised simplex method and how to exploit it. *Comput. Optim. Appl. 32,* 3, 259–283.

Koberstein, A. 2005. The dual simplex method, techniques for a fast and stable implementation. Ph.D. thesis, Universität Paderborn, Germany.

Luce, R. 2007. Analysis of the linear systems of equations arising in the simplex method. M.S. thesis, Technische Universität Berlin, Institut für Mathematik.

Markowitz, H. M. 1957. The elimination form of the inverse and its application to linear programming. *Management Sci. 3*, 255–269.

Maros, I. 2003. *Computational techniques of the simplex method.* International Series in Operations Research & Management Science, 61. Kluwer Academic Publishers, Boston, MA. With a foreword by András Prékopa.

Orchard-Hays, W. 1968. *Advanced Linear-programming Computing Techniques.* McGraw-Hill, Chapter 4, 73–82.

Schenk, O. and Gärtner, K. 2004. Solving unsymmetric sparse systems of linear equations with PARDISO. *Journal of Future Generation Computer Systems 20,* 3, 475–487.

Schenk, O. and Gärtner, K. 2006. On fast factorization pivoting methods for symmetric indefinite systems. *Elec. Trans. Numer. Anal 23*, 158–179.

Schenk, O., Gärtner, K., and Fichtner, W. 2000. Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors. *BIT 40,* 1, 158–176.

Schenk, O., Wächter, A., and Hagemann, M. 2007. Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization. *Comput. Optim. Appl. 36,* 2-3, 321–341.

Suhl, L. M. and Suhl, U. H. 1990. Computing sparse LU factorizations for large-scale linear programming bases. *ORSA J. Compu. 2,* 4, 325–335.

Suhl, L. M. and Suhl, U. H. 1993. A fast *LU* update for linear programming. *Ann. Oper. Res. 43,* 1-4, 33–47.

Tomlin, J. 1972. Pivoting for size and sparsity in linear programming inversion routes. *J. Inst. Math. Appl. 10*, 289–295.

Wunderling, R. 1996. Paralleler und objektorientierter Simplex-Algorithmus. Ph.D. thesis, Technische Universität Berlin.

Yannakakis, M. 1981. Computing the minimum fill-in is NP-complete. *SIAM J. Algebraic Discrete Methods 2,* 1, 77–79.