# Randomized Combinatorial Algorithms for Linear Programming when the Dimension is Moderately High

M. Pellegrini
Istituto di Matematica Computazionale del CNR*
pellegrini@imc.pi.cnr.it

September 27, 2000

## Abstract

In the last decade researchers in computational geometry have produced a series of algorithms for linear programming, based on a randomized combinatorial approach, which are tuned for linear programs where the number of variables $d$ is small compared to the number $n$ of constraints, although not so small to be considered a constant of the problem. One natural question is how practical are these algorithms for classes of LP instances not necessarily derived from problems in computational geometry. In this paper, building within the randomized combinatorial approach, we propose two algorithms for linear programming and we give evidence of their empirical running times on several classes of randomly generated instances.

Comparisons with state of the art free software (lp_solve) and state of the art commercial software (Cplex) lead to the conclusion that the randomized combinatorial approach for systems with $n \approx d$, at the present state of our research, can be competitive for dense systems and for sparse systems where a large fraction of the constraints are equalities.

We also consider the case of dense systems where $n >> d$, which is typical in instances from computational geometry problems, for which we improve upon recent results of Gärtner and Schönherr [13].

*Area della Ricerca di San Cataldo, 56100 Pisa ITALY

## 1 Introduction

Algorithmics for Linear Programming (LP) is a fascinating research area that has attracted for many years sustained efforts from researchers in operating research and design and analysis of algorithms. This is due in part to the practical fall-outs (Dantzig gives a full account of the logistic problems stimulating the invention of the the simplex algorithm in the late 1940's [8]) and in part to intriguing and deep connections of LP with several branches of mathematics. We refer to [20] for an historical account up to the late 80's, and to [4] for a survey of recent developments. We will discuss mostly the LP problem with a geometric terminology and we will sometime refer to standard concepts and terminology borrowed from the simplex algorithm [6] assuming reader's familiarity, without elaborating much on them.

In this paper we concentrate our attention on a class of algorithms developed by Seidel [21], Clarkson [7], Sharir and Welzl [22], Matoušek, Sharir and Welzl [17] and Kalai [16]. These algorithms attack the linear programming problem following a randomized and combinatorial approach that has been applied successfully to many problems in computational geometry (see e.g. [19])[1]. In computational geometry it is often assumed that the dimension of the ambient space (in our case the number $d$ of *variables* in the formulation of the linear program) is small compared to the number of geomet-

[1]Every randomized algorithm calls for its de-randomized partner: a de-randomized counterpart of Clarkson's method has been proposed in [5].

ric entities involved (in our case the number $n$ of linear inequalities and equalities, generically called *constraints*) so that in first approximation the dimension $d$ is treated as a constant. This approach has lead to the celebrated result of Megiddo [18] on linear programming in linear time in fixed dimension. When the dimension of the problem (e.g the number of variables) is not assumed as a constant the worst case analysis of the above mentioned algorithms exhibit an exponential upper bound on the number of steps, except the algorithms in [17] and [16] which are sub-exponential (i.e. they are exponential in $\sqrt{d}$, instead of $d$).

The purpose of this paper is to propose two randomized combinatorial algorithms for linear programming which are based on mixing in a novel way ingredients already present in [21],[7], [22], [17] and [16]. We do not give a complexity analysis, instead we gather evidence on the empirical running time and we make comparisons with state of the art freely available LP software (namely, lp_solve 3.0 of Michel Berkelaar), with state of the art commercial software (CPLEX 6.6.1), and with published empirical results.

In the randomized combinatorial approach to LP the combinatorial properties are more important than metric or algebraic properties. For example, given a point (the partial solution) and a halfspace (a constraint) the fact that the point is or is not included in the halfspace is more important than the distance of the point to the hyperplane supporting the halfspace, or whether that hyperplane has a sparse algebraic representation as a linear polynomial.

The combinatorial approach, very much like in the RISC philosophy, wants to build efficient solution form a very restricted base. If the purpose is to prove, hopefully polynomial, asymptotic bounds this makes perfect sense, however, form an empirical point of view, it is a legitimate question to ask whether practically such a restricted algorithm can be competitive with other traditional approaches that seem to exploit much more information.

In order to answer affirmatively to this question it is not necessary to beat the competing algorithms on every possible input, it is sufficient for the moment to show that there are significant classes of LP instances for which the randomized approach is not at a disadvantage.

Much work along a similar line, for the case when $n \gg d$, has been done by Gärtner [9],[11],[12], Gärtner and Welzl [14], [15], Gärtner and Schönherr [13].

**What is new?** We describe two algorithms (from now on christened LPI and LPII, for lack of better names) for linear programming which combine ideas taken from two algorithms of Clarkson [7] and one from Sharir and Welzl [22]. One of these algorithms, namely LPII, has similarities as well as dissimilarities with one of Gärtner and Welzl described in [15], on which we will comment later. The novelty of the new algorithms is mainly in the way we play with the *size*, $r$, of the random sample. Clarkson uses a size $r = d\sqrt{n}$, while Matoušek, Sharir and Welzl use $r = n-1$. Here we use $r = \sqrt{n}$. With our choice, the sample size in independent of $d$ and it is always smaller than the size of the sampled set. A first advantage is that the recursion can go down to sets of size 1 so that auxiliary traditional algorithms for the bottom case are avoided. From a broader viewpoint, we try to change the perspective and to suggest that the good empirical performance of a particular size rule might be a clue also for further investigations in algorithmic complexity.

**Input Generation and Average Running Time.** The issue of the average behaviour of the simplex algorithm, under a variety of pivoting rules, has been treated by several authors in the 80's (to cite a few results: [1], [2] and [3]). In the three articles mentioned above the probabilistic model on which the averaging is defined gives equal probability to any combination of inequalities for a given numerical input. Geometrically speaking this means that for a fixed arrangement of hyperplanes, *every cell* in the arrangement has equal probability of being the feasibility set. In our setting we have two different random processes involved. The first random process produces instances of LP problems, the second random process is used internally by the algorithms to decide on which constraint the *pivoting* should be done. Our input data is generated so that for any possible arrangement of hyperplanes that is generated *only one* cell of the arrangement is the feasibility set of the corresponding LP problem. Thus the averages we compute and those considered in the above mentioned articles are of a different nature. This should be sufficient to dispel the suspect that the good behaviour of the presented algorithms might be a trivial consequence of the generally good behaviour of the simplex-like methods "on average".

**Experimental set up.** We consider three classes of LP instances. The first class (I) consists of LP instances where $n >> d$ and the constraints are 100% dense, which is typical for LP obtained from problems in computational geometry. The second class (II) consists of LP instances where $n \approx d$ and the LP has a mix of equalities and inequalities (both dense and sparse). The third class (III) consist of instances of LP where $n \approx d$ and the LP has only inequalities (both dense and sparse).

**Discussion of results.** The experimental set up is described in more detail in section 5 and the raw data are given in section 8. Here we briefly summarize the conclusions drawn from the experiments. In all experiments with $n \approx d$ our algorithm LPI has consistently better performance than lp_solve for inputs large enough.

Cplex is a much more expensive and sophisticated software and it is quite good in exploiting input sparsity. In contrast our treatment of sparsity is still embryonic. This fact explains rather well results of Table 8 where Cplex out-performs LPI by a factor between 1.5 and 2. A confirmation of this interpretation is provided in Tables 6 and 7 where dense instances strip Cplex of its advantage in handling sparsity and LPI matches Cplex.

We have a different picture when the input LP instances contain a fair number of equalities. LPI seems to be better than Cplex in treating equalities so that for dense systems with equalities the two factors are both at LPI's advantage and LPI can take the lead (Tables 3 and 4). When we consider sparse inputs with equalities the two effects balance each other so that in Table 5 LPI and Cplex tie.

Naturally this exercise does not automatically imply that our implementation of LPI is in every respect "better" than lp_solve or of cplex in some cases, since several issues in robustness[2] and sparsity have not been addressed yet. The safest conclusion we feel we can draw, in contrast with standard intuition and the opinion of the usual skeptic, is that these simplex-like algorithms using information sparingly have nonetheless a fair chance of competing on a level field with traditional simplex-like algorithms which strive to collect and use much more information about the problem.

LPII is suitable for dense problems with $n >> d$ and in Tables 1and 2 we extend the maximum range of values of $d$ from 20 to 200 compared to similar experiments in [13].

[2]LP benchmarks in the NETLIB collection are rather tough for the numerical stability of LP algorithms.

## 2 Brief formulation of the problem

We consider the following formulation of the LP problem: Find the point $x \in R^d$ minimizing the linear objective function $c \cdot x$ with the constraints:

$$x_j \geq 0 \quad \forall j = 1, .., d \text{ (Positivity constraints)}$$

and

$$\sum_{j=1}^{d} a_{ij} x_j \geq b_i \quad \forall i = 1, .., n \text{ (Matrix constraints)}$$

for a given $(n \times d)$ matrix $A = [a_{ij}]$, $d$-vector $c$ and $n$-vector $b$. In geometric terms each constraint (positivity and matrix constraints alike) is an *halfspace* $h_k$ in $R^d$, for $k = 1, .., d+n$. The set of points satisfying all the constraints is the intersection of all the halfspaces $P = \bigcap h_k$, which is a polyhedron in $R^d$. The locus of points satisfying a constraint with equality is an *hyperplane*. We denote with $H_+$ the set of halfspaces from the positivity constraints, with $H_M$ the set of halfspaces from the matrix constraints and with $H$ their union.

We are looking for the point in $P$ minimizing the linear objective function $c \cdot x$. If $P$ is empty the problem is called *unfeasible*. If $P$ is not empty but there is no minimum of $c \cdot x$ in $P$, the problem is called *unbounded*. When the problem is *feasible* and *bounded* the minimum is attained at a vertex. If there is more than one vertex attaining the minimum we will choose the lexicographically smallest minimum vertex so that the solution is unique.

A *basis* $B$ is a set of $d$ constraints in $R^d$ such that the vertex $p(B)$ of the polyhedral cone formed by the intersection of the $d$ halfspaces is the minimum of the objective function for points of the cone. Thus a basis is characterized by a set of constraints $B$, a polyhedral cone $C(B)$, a point $p(B)$ which is the vertex of the corresponding cone $C(B)$ and a value $v(B) = c \cdot p(B)$ which is the value of the objective function at the vertex of the cone. Finally an halfspace $h$ *violates* a basis $B$ if $h \notin B$ and $p(B) \notin h$.

Solving the LP problem above for a set of halfspaces $H$ corresponds to finding a basis $B \subset H$ such that no constraint in $H \setminus B$ violates $B$. This is seen easily with the following argument. For a basis $B$ with the above properties, $p(B)$ is in the feasible polyhedron $P$ and, moreover, $P \subset C(B)$. Suppose, by contradiction, that there is a point $q \neq v(B)$ in $P$ with a value of the objective function smaller than $v(B)$. Since such point is also in $C(B)$, we have a contradiction with the minimality of $v(B)$.

Without loss of generality we consider an objective function $c$ such that $c_j \geq 0$ for each $j = 1..d$. so that the set of positive constraints $H_+$ is a basis, with apex at the origin and objective value zero[3].

### 2.1 Subroutine Newbasis

The basis of the recursion in our algorithm is a procedure $newbasis(B, \{h\})$ taking as input a basis $B$ and a violating constraint $h$, and produces a new basis $B' \subset (B \cup \{h\})$, such that the constraint excluded $(B \cup \{h\}) \setminus B'$ is not violating for $B'$. Always $v(B') > v(B)$. With a convenient representation based on the edges of the cone $C(B)$ we can compute in time $O(d^2)$ the new basis $B'$ and the representation for the new cone $C(B')$. Newbasis correspond to the pivoting operation in a simplex algorithm.

## 3 New Algorithms

### 3.1 Algorithm LPI

Algorithm LPI takes as input the following parameters: a basis $B$ and a set $H$ of constraints not in the basis. We maintain the invariant: $B \cap H = \emptyset$ on all calls of the algorithm[4]. The cardinality of the input sets is $|B| = d$, $|H| = n$. The first call is lpI($H_+$, $H_M$) the positivity constraints are the first basis, the matrix constraints the set of extra constraints.

lpI ($B$,$H$)
{
$V = \{h \in H \mid violates(h, B)\}$
if ($|H| = 0$) or ($|V| = 0$)
  then return $B$;
if ($|H| = 1$) and ($|V| = 1$)
  then return newbasis($B$, $H$);
$r = \lfloor f * \sqrt{|V|} \rfloor$;
$R$ = randomsubset($V$, $r$);
$B'$ = lpI($B$, $R$);
return lpI($B'$, $(B \cup H) \setminus B'$);
}

Algorithm LPI compute the set $V$ of violators in $H$ and then determines a random sample $R$ of the set of violators $V$. It solves recursively the linear program given by the current basis $B$ and the sample $R$ to produce a new basis $B'$ with which a tail recursion is started. The real number $f$ is a parameter to be tuned. In our experiments we have chosen $f$ between 1.0 and 2.0. Moreover $f$ should be chosen so to have a value of $r$ satisfying $1 \leq r < |V| \leq |H|$.

Termination is easy to prove by induction on the size of the input and on the value of the current basis. The first recursive call is on a set of smaller size, the second recursive call is on a basis with higher value. The algorithm terminates because the value of the basis can only increase during the lifetime of the algorithm and there is a finite number of possible basis, so eventually the basis with maximum value is attained.

### 3.2 Algorithm LPII

We use the same convention and terminology of LPI, as before the first call is lpII($H_+$, $H_M$), where the positivity constraints are the first basis, the matrix constraints the set of extra constraints.

lpII ($B$,$H$)
  {
  if ($|H| = 0$) then return $B$;
  if ($|H| = 1$) then return newbasis($B$,$H$);
  $r = \lfloor f * \sqrt{|H|} \rfloor$;
  $R$ = randomsubset($H$,$r$);
  $B'$= lpII($B$, $R$);
  $V = \{h \in (B \cup H) \setminus B' | violates(h, B')\}$;
  while ($|V| > 0$)
    {
    $B'$ = lpII($B'$,$V$);
    $V = \{h \in (B \cup H) \setminus B' | violates(h, B')\}$;

[3] An easy way to weave this restriction is by introducing an extra variable forced to be equal to the initial linear objective function and setting such new variable as the new objective.

[4] This is a different convention from [17].

};
return $B'$;
}

Algorithm LPII is similar in spirit to algorithm 5.1 in [15], there are however two differences. In LPII the recursive call within the while loop is on the set of violators with respect to the current basis, while in Alg5.1 of [15] it is with respect to an accumulator set that is the union of the violator on all previous iterations. In order to attain a bound $O(d\sqrt{n})$ on the maximum size of the accumulator set according to theorem 5.5 in [15] the sample size has to be chosen to be $O(d\sqrt{n})$. This limits the validity of the sampling size rule proposed in [15] and effectively used in [13] and [11] to inputs where $d \ll \sqrt{n}$. In our algorithm we avoid the use of an accumulator set and our sampling size rule based only on $|H|$ does not place any restriction to the relative values of $n$ and $d$. For this reason the recursive calls to LPII can handle sets of decreasing size up to the situation that we can handle directly a recursive call with the pivoting step procedure Newbasis.

### 3.3 Comparing LPI and LPII

In LPI we find the violators first, we sample only among the violators and then we call the procedure recursively on the sampled violators. The basis returned is used as starting point of a tail recursion. In LPII we sample first, we call the procedure recursively on the sample set and then we find the violators. The returned basis is used as stating point of a loop where we continue to iterate on the set of violators induced by the iteratively re-computed basis.

LPI is more effective (empirically) when the set of violators $V$ is relatively large with respect to $H$. Conversely, LPII is more effective (empirically) when the set of violators $V$ is relatively small with respect to $H$. Algorithm LPI is faster than algorithm LPII when $n \approx d$, while LPII is faster than LPI when $n \gg d$.

## 4 Issues

### 4.1 Numerical stability

All arithmetic internal operations of our algorithms use floating point arithmetic (C double). This choice is justified since the objective of this paper is testing empirically the running time complexity. Numerical stability has been pursued only up to the point of allowing testing with large inputs. As a consequence occasional failure, as long as it is detected, is acceptable. Occasional numerical failures occurred during the experiments of class II and III have been detected and recovered, but not accounted for in the timing. No numerical failure has been detected in experiments for class I.

The use of floating point arithmetic rises the issue of the tolerance acceptable in the final solution. In checking the final result we have considered acceptable a vertex that would satisfy the constraints in the basis and does not violate any other constraint within a tolerance of $10^{-7}$. These two tests are sufficient to conclude that the output vertex is in the feasibility polyhedron but, if the solution is not known otherwise, do not guarantee minimality of the objective function. However, since different runs of a randomized algorithm over the same input involve different sequences of arithmetic operations, a consensus within the prescribed tolerance over 10 different random runs is take as sufficient evidence that numerical instabilities have not occurred in the determination of the minimum.

### 4.2 Uniform and biased random sampling

In [7] are shown two algorithms for linear programming, one based o uniform random sampling and one based on the technique of iterative reweighting coupled with biased sampling. Thus have considered three main variants for LPI:

- U.W. Uniformly weighted random sampling: The set $R$ is taken uniformly at random in the set $V$ of violators of $B$ in $H$.
- L.W. Linearly weighted random sampling: We associate to each $h$ in $H \cup B$ the number $w(h)$ of times $h$ has been found to be a violator. The random subset $R$ of $V$ is chosen so that the probability of each element in $V$ is proportional to $w(h)$.
- E. W. Exponentially weighted random sampling. We compute $w(h)$ as above but we use as weights in the sample the values $x^{w(h)}$ for a real $x > 1$.

Of these variants L.W. has the best performance and all experimental data for LPI are obtained with linear weighting.

### 4.3 LP-type and Abstract Optimization problems

Combinatorial randomized algorithms for Linear programming have been shown to be templates for a larger classes of problems known as LP-type problems [22] and Abstract Optimization Problems (AOP) [10, 9]. Provided certain combinatorial properties are satisfied by a specific problem one can get the an optimization algorithm for that specific problem by supplying an implementation of the pivoting step which is the only problem-specific part. A well studied LP-type problem is that of the minimum enclosing ball [23] for a set of points in $d$-space. In our experiments we concentrate on the LP problem and it remains an open direction of research to determine whether the good behaviour in higher dimension can benefit also those larger classes of problems.

## 5 Experimental set up

**Implementation.** Algorithms LPI and LPII have been implemented in C, compiled with -O4 option of gcc. Experiments with LPI, LPII, and lp_solve are run on an AMD Athlon 800 MHz, 528Mb RAM under Linux. Experiments with Cplex for $d \leq 300$ have been made with MPL/Cplex 4.11 on a Pentium II, 400 MHz clock, 256Mb RAM memory, under Windows; experiments for $d > 300$ have been made with Cplex 6.6.1 on a Pentium II 400 MHz, 256M RAM memory under Linux. Cplex has been run with default parameters and a primal simplex solving strategy has been chosen.

**Input generation.** In our experiments we generate the input data $(A, b, c)$ as follows. The components of $A$ ad $b$ and $c$ are floating point numbers chosen uniformly at random in the interval $[2^{-1}, 2]$, and the components of the constant vector $b$ are then shifted to ensure the feasibility set is not empty. In tables 6 7 and 8 the coefficients of $A$ are chosen uniformly at random in the interval $[-2^{-1}, 2]$.

**Performance evaluation.** For each value of $n$ and $d$ in tables 1 and 2 is shown the average number of calls of the pivoting routine newbasis, the average total time in seconds, and the average time spent for pivoting, over 10 different inputs of given size and 10 different runs of LPII with the same input. For all other table the performance of LPI is the average over 10 runs on a single input of the specified size.

**Input Class I**. In Table 1 we have considered linear systems with $n = 20,000$, $d$ variables in the range [20..200], and $v = d$ non-zero coefficients per equation. In Table 2 we have considered linear systems with $n = 200,000$, $d$ variables in the range [20..100], and $v = d$ non-zero coefficients per equation. The purpose of the experiment is to show that simple randomized pivoting rules can achieve good empirical running times when $n \gg d$. It is instructive to compare our results with those obtained by Gärtner and Schönherr [13] (Table 13, algorithm QP with partial pricing, floating point (double) arithmetic) for $d = 22$, $n = 20,000$ a time of 0.677 seconds is attained, and for $d = 22$, $n = 200,000$ a time of 3.6 seconds (Table 14). According to Gärtner and Schönherr larger values of $d$ lead to severe numerical problems. We attain in Tables 1 and 2 comparable times for problems of that size[5], but we could greatly increase the range of allowable values for $d$, while keeping the running times within acceptable limits.

**Input Class II**. In Tables 3, 4 and 5 we have considered linear systems with a total of $n = d$ constraints and variables in the range $[100, 600]$. In each input 34% of constraints are equalities, 66% are inequalities. Density in 100% in the first table, 50% in the second and 5% in the third. LPI has better performance than lp_solve in all three tables. LPI has better performance than Cplex in two tables with density 100% and 50%. For density 5% LPI and Cplex have comparable performance.

**Input Class III**. In Tables 6, 7 and 8 the setting is as in class II but each instance is formed only of inequalities. LPI has better performance than lp_solve in all three tables. LPI has similar performance than Cplex in tables with density 100%. LPI starts loosing ground for density 50%. For density 5% Cplex is clearly faster than LPI.

[5] A minimum enclosing annulus problem on $n$ points in dimension $d$ corresponds to a LP problem in $2n$ constraints and dimension $d + 2$.

## 6 Conclusions and Open problems

The main idea of this paper is that the size of the sample matters quite a lot for the efficiency of randomized LP algorithms and preliminary experimental results support this intuition. This paper reports on on-going research and it does not claim to have settled all related issues nor to be any definite answer to the problem. Other important components of a successful LP code are resilience to numerical instabilities and handling of sparse inputs. Both issues have been just only touched in the present research and I suspect that there is much scope for improvement here.

## 7 Acknowledgments

Boris Aronov has given me valuable support in early stages of this research. Ken Clarkson and Emo Welzl have kindly answered many questions on randomized LP problems. Antonio Frangioni has kindly given me access to Cplex.

## References

[1] I. Adler, R. M. Karp, and R. Shamir. A simplex variant solving an $m$ x $d$ linear program in $O(min(m^2, d^2))$ expected number of pivot steps. *J. Complexity*, 3:372–387, 1987.

[2] K.-H. Borgwardt. Some distribution-independent results about the asymptotic order of the average number of pivot steps of the simplex method. *Mathematics of Operations Research*, 7:441–462, 1982.

[3] K.-H. Borgwardt. *The Simplex Method. A Probabilistic Analysis*, volume 1 of *Algorithms and Combinatorics; Study and Research Texts.* Springer-Verlag, Berlin, 1987.

[4] V. Chandru and M. R. Rao. Linear programming. In *M.J. Atallah (Editor-in-Chief), Algorithms and Theory of Computation Handbook.* CRC Press, 1999.

[5] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1993.

[6] V. Chvátal. *Linear Programming.* W. H. Freeman and Company, New York, 1983.

[7] K. L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM*, 42(2):488–499, March 1995.

[8] G. B. Dantzig. *Linear Programming and Extensions.* Princeton Univ. Press, Princeton, NJ, 1962.

[9] B. Gärtner. *Randomized optimization by Simplex-type methods.* PhD thesis, Institut für Informatik, Freie Universität Berlin, 1995.

[10] B. Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM Journal on Computing*, 24(5):1018–1035, October 1995.

[11] B. Gärtner. Exact arithmetic at low cost – A case study in linear programming. *CGTA: Computational Geometry: Theory and Applications*, 13, 1999.

[12] B. Gärtner. Fast and robust smallest enclosing balls. *Lecture Notes in Computer Science*, 1643:325–338, 1999. Proceedings of the 7th Annual European Symposium on Algorithms.

[13] B. Gärtner and S. Schönherr. An efficient, exact and generic quadratic programming solver for geometric optimization. To appear in the 16th ACM Symposium on Computational Geometry, June 2000.

[14] B. Gärtner and E. Welzl. Linear programming—randomization and abstract frameworks. In *13th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1046 of *lncs*, pages 669–687, Grenoble, France, 22–24 February 1996. Springer.

[15] B. Gärtner and E. Welzl. Random sampling in geometric optimization: New insights and applications. To appear in the 16th ACM Symposium on Computational Geometry, June 2000.

[16] G. Kalai. A subexponential randomized simplex algorithm. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 475–482, Victoria, B.C., Canada, May 1992.

[17] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, October/November 1996.

[18] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, January 1984.

[19] K. Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms.* Prentice-Hall, Englewood Cliffs, NJ, 1994.

[20] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1987.

[21] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comput. Geom.*, 6:423–434, 1991. See also *6th Symp. Comp. Geom.*, 1990, pp. 211–215.

[22] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In Alain Finkel and Matthias Jantzen, editors, *Proceedings of Symposion on Theoretical Aspects of Computer Science (STACS '92)*, volume 577 of *LNCS*, pages 569–588, Berlin, Germany, February 1992. Springer.

[23] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *H. Maurer (Ed.), New Results and New Trends in Computer Science, LNCS 555.* 1991.

# 8 Tables

Enclosed in () is given the average total time spent in performing the pivoting operations. Time data for Cplex, lp_solve and LPI, LPII exclude the time for file parsing, pre-processing and input generation.

| n | d | LPII # calls | LPII time |
|---|---|---|---|
| 20,000 | 20 | 156.25 | 0.14 ( 0.00 ) |
| 20,000 | 40 | 232.75 | 0.33 ( 0.01 ) |
| 20,000 | 60 | 327.84 | 0.64 ( 0.03 ) |
| 20,000 | 80 | 425.04 | 0.98 ( 0.07 ) |
| 20,000 | 100 | 584.75 | 1.47 ( 0.14 ) |
| 20,000 | 200 | 1028.88 | 4.34 ( 1.01 ) |

Figure 1: Experiment with 100% density, averages over same input (10 iterations), and same size (10 iterations). $f = 1.4$. Time in seconds.

| n | d | LPII # calls | LPII time |
|---|---|---|---|
| 200,000 | 20 | 216.22 | 1.17 ( 0.00 ) |
| 200,000 | 40 | 353.29 | 2.81 ( 0.02 ) |
| 200,000 | 60 | 495.52 | 4.24 ( 0.06 ) |
| 200,000 | 80 | 697.17 | 6.56 ( 0.12 ) |
| 200,000 | 100 | 805.23 | 8.92 ( 0.23 ) |

Figure 2: Experiment with 100% density, averages over same input (10 iterations), and same size (10 iterations). $f = 1.4$. Time in seconds.

| n | d | LPI # calls | LPI time | lp_solve | cplex |
|---|---|---|---|---|---|
| 100 | 100 | 226.90 | 0.07 ( 0.04 ) | 0.08 | 0.15 |
| 200 | 200 | 858.70 | 1.03 ( 0.69 ) | 2.32 | 2.45 |
| 300 | 300 | 1697.20 | 5.18 ( 3.90 ) | 14.51 | 11.72 |
| 400 | 400 | 2853.10 | 16.86 ( 13.40 ) | 57.72 | 30.86 |
| 500 | 500 | 4433.10 | 42.68 ( 35.59 ) | 197.01 | 66.50 |
| 600 | 600 | 5490.40 | 75.18 ( 64.07 ) | 393.32 | 130.42 |

Figure 3: Experiment with 100% density, 34% equalities, 66% inequalities, averages over same input (10 iterations). $f = 1.4$. Time in seconds.

| n | d | LPI # calls | LPI time | lp_solve | cplex |
|---|---|---|---|---|---|
| 100 | 100 | 264.60 | 0.07 ( 0.04 ) | 0.05 | 0.16 |
| 200 | 200 | 853.60 | 0.83 ( 0.60 ) | 1.34 | 1.17 |
| 300 | 300 | 1864.10 | 4.67 ( 3.64 ) | 12.62 | 10.73 |
| 400 | 400 | 2813.10 | 13.91 ( 11.50 ) | 54.05 | 27.24 |
| 500 | 500 | 4047.00 | 33.11 ( 28.62 ) | 135.80 | 55.69 |
| 600 | 600 | 5918.00 | 70.53 ( 62.34 ) | 403.86 | 119.39 |

Figure 4: Experiment with 50% density, 34% equalities, 66% inequalities, averages over same input (10 iterations), and same size (10 iterations). $f = 1.4$. Time in seconds.

| n | d | LPI # calls | LPI time | lp_solve | cplex |
|---|---|---|---|---|---|
| 100 | 100 | 171.20 | 0.03 ( 0.01 ) | 0.01 | 0.03 |
| 200 | 200 | 695.10 | 0.49 ( 0.37 ) | 0.28 | 0.28 |
| 300 | 300 | 1387.90 | 2.58 ( 2.14 ) | 2.87 | 2.25 |
| 400 | 400 | 2655.90 | 10.17 ( 8.93 ) | 15.53 | 6.69 |
| 500 | 500 | 3878.20 | 24.51 ( 21.91 ) | 62.30 | 22.29 |
| 600 | 600 | 5227.30 | 48.00 ( 43.68 ) | 150.57 | 44.71 |

Figure 5: Experiment with 5% density, 34% equalities, 66% inequalities, averages over same input (10 iterations). $f = 1.4$. Time in seconds.

| n | d | LPI # calls | LPI time | lp_solve | cplex |
|---|---|---|---|---|---|
| 100 | 100 | 338.50 | 0.14 ( 0.07 ) | 0.09 | 0.17 |
| 200 | 200 | 803.60 | 1.39 ( 0.93 ) | 1.16 | 2.36 |
| 300 | 300 | 1439.50 | 6.68 ( 5.03 ) | 9.38 | 10.79 |
| 400 | 400 | 2214.60 | 19.28 ( 15.35 ) | 21.9 | 19.33 |
| 500 | 500 | 2799.60 | 37.20 ( 30.55 ) | 46.08 | 37.84 |
| 600 | 600 | 3846.70 | 72.81 ( 61.10 ) | 98.14 | 67.05 |

Figure 6: Experiment with 100% density, 100% inequalities, averages over same input (10 iterations). $f = 1.4$. Time in seconds.

| n | d | LPI # calls | LPI time | lp_solve | cplex |
|---|---|---|---|---|---|
| 100 | 100 | 456.60 | 0.15 ( 0.09 ) | 0.15 | 0.12 |
| 200 | 200 | 1158.10 | 1.63 ( 1.23 ) | 1.45 | 1.25 |
| 300 | 300 | 2029.60 | 7.66 ( 6.26 ) | 8.54 | 7.81 |
| 400 | 400 | 3174.30 | 23.07 ( 19.83 ) | 31.09 | 15.09 |
| 500 | 500 | 4247.90 | 48.01 ( 41.85 ) | 68.76 | 37.86 |
| 600 | 600 | 5378.00 | 85.63 ( 76.12 ) | 136.54 | 64.84 |

Figure 7: Experiment with 50% density, 100% inequalities, averages over same input (10 iterations). $f = 1.4$. Time in seconds.

| n | d | LPI # calls | LPI time | lp_solve | cplex |
|---|---|---|---|---|---|
| 100 | 100 | 204.00 | 0.05 ( 0.03 ) | 0.02 | 0.02 |
| 200 | 200 | 1182.40 | 1.31 ( 1.08 ) | 0.8 | 0.23 |
| 300 | 300 | 2509.70 | 7.39 ( 6.44 ) | 8.27 | 1.50 |
| 400 | 400 | 4255.90 | 24.60 ( 22.31 ) | 31.86 | 4.99 |
| 500 | 500 | 6513.90 | 60.16 ( 55.73 ) | 118.98 | 14.34 |
| 600 | 600 | 8674.40 | 113.66 ( 106.20 ) | 287.40 | 33.22 |

Figure 8: Experiment with 5% density, 100% inequalities, averages over same input (10 iterations). $f = 1.4$. Time in seconds.