Available online at www.sciencedirect.com
ScienceDirect
Operations Research Letters 35 (2007) 693–699
Operations Research Letters
www.elsevier.com/locate/orl

Exact solutions to linear programming problems

David L. Applegate[a], William Cook[b], Sanjeeb Dash[c], Daniel G. Espinoza[d,*]

[a]*AT&T Labs, Research, 180 PARK AVE, P.O. BOX 971, Florham Park, NJ 07932-0971, USA*
[b]*School of Industrial & Systems Engineering, Georgia Institute of Technology, 765 Ferst Drive NW., Atlanta GA, 30332, USA*
[c]*IBM T. J. Watson Research Center, 1101 Kitchawan Road, Route 134, Yorktown Heights, NY 10598, USA*
[d]*Departamento de Ingeniería Industrial, Universidad de Chile, Av. República 701, Santiago 837-0439, Chile*

Received 17 October 2006; accepted 28 December 2006
Available online 21 February 2007

Abstract

The use of floating-point calculations limits the accuracy of solutions obtained by standard LP software. We present a simplex-based algorithm that returns exact rational solutions, taking advantage of the speed of floating-point calculations and attempting to minimize the operations performed in rational arithmetic. Extensive computational results are presented.

MSC: 90C05; 90C49

Keywords: Linear programming; Simplex algorithm; Rational arithmetic

1. Introduction

Standard linear programming (LP) solvers can report different "optimal" objective values for the identical problem on different computer architectures. This inconsistency is primarily due to the use of floating-point numbers in LP software. Floating-point computations can lead to nontrivial errors in the context of LU factorization or Cholesky factorization—operations used by most solvers.

Although good approximate solutions are satisfactory in many LP applications, there are scenarios that require exact values, such as when LP is used to compute theoretical bounds for various problems. Moreover, in some cases industrial customers request exact solutions (Zonghao Gu, personal communication, 2005). This interest in accurate LP results has prompted recent studies by Gärtner [9], Jansson [14], Dhiflaoui et al. [5] and Koch [16].

In this paper we present an implementation of the simplex algorithm that provides exact solutions to LP instances, while attempting to minimize the arithmetic operations performed using rational arithmetic. We report test results for benchmark LP instances and for computations of exact solutions for the subtour

* Corresponding author.
E-mail addresses: david@research.att.com (D.L. Applegate), bico@isye.gatech.edu (W. Cook), sanjeebd@us.ibm.com (S. Dash), daespino@dii.uchile.cl (D.G. Espinoza).

0167-6377/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.orl.2006.12.010

relaxation of the traveling salesman problem (TSP) and exact solutions to small mixed-integer programming (MIP) problems. Our code [3] is available to the academic community.

2. A first approach

A natural method to obtain exact LP solutions is to implement a solver that computes entirely in rational arithmetic. To achieve this we began with the source code for the QSopt [2] implementation of the simplex algorithm, changed every floating-point type to the rational type provided by the GNU-MP (GMP) library [11], and changed every operation in the original code to use GMP operations. (See [15] for another implementation of the simplex algorithm in full rational arithmetic.)

We tested this rational solver on a set of 170 instances taken from the union of benchmark examples described in Section 4.1. (The selected subset consists of all problems whose MPS input file is no larger than that of the NETLIB problem pilot4.) A summary of these results can be found in Table 1, where the second column reports the number of nonzeros in the constraint matrix plus the number of constraints and variables, the third column gives the number of simplex pivots, and the fourth column gives the running time of the code in seconds. The fifth column gives the average number of bits needed to represent each nonzero entry of the optimal primal and dual solutions, and the last column gives the ratio between the running time of the rational code and the time needed for the original QSopt code. Table 1 shows the details for all problems whose running-time ratio was above 800, and an average for all other instances. All runs were carried out on a Linux workstation with a 2.4 GHz AMD Opteron 250 CPU and with 8 GB of RAM.

The test results show a wide variance in the length of the solution encoding, together with a strong correlation between these lengths and the average time needed in each simplex iteration. These factors lead to highly unpredictable behavior for the overall code, making this naive method impractical for most applications.

3. Using floating-point arithmetic

Dhiflaoui et al. [5] pioneered an alternative approach for obtaining exact LP solutions, using the output of a floating-point solver as a starting point for rational computations. Rather than considering the primal and dual solutions provided by the floating-point computation, Dhiflaoui et al. take the description of the proposed optimal basis and compute the corresponding rational solution. Working with the simplex algorithm as implemented in ILOG CPLEX [13], they found that in many cases the rational solution is indeed optimal, and in other cases a small number of additional pivots in rational arithmetic could be used to

Table 1
Running times for rational solver on 170 small problems

Instance	Size	Pivots	Time (s)	Encoding	Ratio
brandy	3746	205	52.39	425	2619.5
grow7	3914	185	55.13	1294	2756.5
e226	4257	320	27.55	437	1317.5
bandm	4963	297	20.84	642	833.6
forplan	6210	154	6.98	309	807.5
stair	6570	366	653.52	4051	7261.3
danoint	7594	990	289.56	479	1206.5
small002	8791	991	26.38	1380	858.5
small008	8870	1099	36.06	855	876.7
small005	8882	1110	28.60	550	817.9
small007	8908	1102	78.51	771	1121.6
pilot4	9191	1041	5547.03	5606	27,715.2
modszk1	9843	777	458.43	826	2865.2
Others	6858	562	5.31	120	108.4

Table 2
Running times for pilot4

Number Representation	Time (s)
double	0.20
GMP-float (64 bits)	5.57
GMP-float (128 bits)	5.82
GMP-float (256 bits)	8.12
GMP-float (384 bits)	11.37
GMP-float (1,024 bits)	30.59
GMP-rational	5547.03

arrive at an optimal solution. Their tests were carried out on a subset of NETLIB instances having at most 2400 rows. For larger or more numerically difficult examples, however, the rational pivot steps needed to repair a nonoptimal basis can lead to behavior similar to what we saw in the previous section, making the overall process challenging to carry out in practice.

Koch [16] modified this approach to compute optimal solutions for the full set of NETLIB instances. Koch [16] writes the following:

"The current development version of SOPLEX using 10^{-6} as tolerance finds true optimal bases to all instances besides `d2q05c`, `etamacro`, `nesm`, `dfl001`, and `pilot4`. Changing the representation from 64 to 128 bit floating-point arithmetic allows also to solve these cases to optimality".

Thus, rather than attempting to repair a non-optimal basis with rational pivots, Koch recomputed a floating-point solution using greater precision in the floating-point representations. In personal communication, Koch explained that in these calculations he employed the *long double* type provided on a computer architecture that uses 128-bit values.

We extend Koch's methodology with an implementation that dynamically increases the precision of the floating-point computations until we obtain a basis that yields rational primal and dual solutions satisfying the optimality, unboundedness, or infeasibility conditions. The GMP library allows us to perform floating-point calculations with arbitrary (but fixed) precision, and moreover, to adjust this precision at running time.

Motivation for our method can be seen in the results for the pilot4 LP problem given in Table 2, where we display the total solution time for increasing levels of precision. We aim to take advantage of the fast times for small precision to limit the number of pivots required at higher levels. An overview of the method is presented in Algorithm 1. In our code, called *QSopt_ex*, we increase the precision p to 128 bits in the second iteration, and to approximately $1.5p$ in each following round (keeping p a multiple of 32 to align with the typical word size).

Algorithm 1. `Exact_LP_Solver.`
Require: $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$, $A \in \mathbb{Q}^{m\times n}$
1: Start with the best native floating-point precision p (number of bits for floating-point representation).
2: Set $\mathscr{B} \leftarrow \emptyset$.
3: Compute approximations $\bar{c}, \bar{b}, \bar{A}$ of the original input in the current floating-point precision.
4: Solve $\min\{\bar{c}x : \bar{A}x \leqslant \bar{b}\}$ using $\mathscr{B}$ (if it is not empty) as the starting basis.
5: $\mathscr{B} \leftarrow$ ending basis of the simplex algorithm.
6: Test result in rational arithmetic.
7: **if** Test fails **then**
8: Increase precision p
9: **goto** step 3
10: **end if**
11: **return** x^*

Our rational tests for optimality, unboundedness, and infeasibility are based on the QSopt LU factorization routines but implemented with rational arithmetic. This means that every test involves a time-consuming rational factorization of the constraint matrix; an alternative strategy is to employ Wiedemann's method [22] for directly solving the rational systems, as discussed briefly in [5]. In an attempt to avoid the rational solve altogether, we first find rational approximations of the floating-point primal and dual solutions using continued fractions, and test the optimality conditions on these approximations before computing the factorization. This approach works well on easier LP examples, such as those in our TSP subtour tests in Section 4.3, as the rational primal and dual values given by the optimal basis do not have large denominators, and the continued fraction method correctly identifies these values from the floating-point approximations.

4. Computational experience

We describe our computational experiments with four classes of LP problems, testing a range of aspects of the QSopt_ex code.

Table 3
Ratios of QSopt_ex vs. QSopt running times

Set	Alg.	Geometric mean	Total	QSopt (s)
Small	Primal	5.7	237.0	15.8
Small	Dual	5.1	131.8	10.1
Medium	Primal	5.2	6.3	414.6
Medium	Dual	6.9	5.8	361.8
Large	Primal	1.8	2.2	3621.5
Large	Dual	2.4	3.3	2529.1

Table 4
Infeasible LP instances

Instance	QSopt (s)	QSopt_ex (s)	Precision
ceria3d	0.8	1.5	Double
cplex1	1.0	2.1	Double
cplex2	0.1	1.1	128 bit
gosh	4.3	6.4	Double
gran	5561.3	714.3	128 bit
greenbea	2.8	5.4	Double
klein3	0.7	1.3	Double

4.1. Benchmark LP instances

We begin with a study of 625 instances taken from the GAMS World library [8], including the MIPLIB, Miscellaneous, NETLIB, Problematic, and Stochastic collections. A full listing of the problems and our complete results are available in [6].

We do not report on problems where the sum of the running times of the primal and dual simplex algorithm for both the exact LP solver and the original QSopt code is less than 1 s; this leaves 364 problems. Furthermore, problems nug20, nug30, cont11, cont1, cont11_l, and cont1_l could not be solved in less than five days of running time and were removed from the overall list of problems that we report.

In Table 4 the running times are given for the seven infeasible LP problems in our test set. The problems were solved with the primal simplex algorithm; the final column reports the floating-point precision p that was needed to obtain a rational certificate of infeasibility.

The feasible LP problems are split into three categories, those with less than 1000 constraints are considered small (66 instances), those with 1000–10,000 constraints are considered of medium size (214 instances), and those with over 10,000 constraints are considered large (83 instances). Table 3 presents a summary of our results for these instances, giving the geometric mean of the running-time ratios for QSopt_ex vs. QSopt and the ratio of the total running times (for solving all instances in a given category) for the two codes; the final column gives the mean number of seconds for QSopt.

Note that the improvement in the running-time ratios for larger instances is due in part to the fact that the number of operations in standard double arithmetic grows larger compared to the number of operations in rational arithmetic as the problems increase in size.

One factor contributing to the increase in running time (over the standard QSopt code) for some of the most difficult LP instances is that our double-precision port of QSopt is not as stable as the original code, resulting in failures at the starting level of precision. In these cases, the final basis is often numerically ill behaved and we are thus forced to restart our algorithm from scratch with increased floating-point precision, at a large computational expense. (see also Table 4)

Of all the optimal solutions collected during our experiments, the largest encoding occurred in the problem stat96v3, requiring on average 64,000 bits to represent each solution coefficient. Fortunately, most solutions have much smaller representations; 80% of the problems required less than 256 bits to exactly represent their optimal solution.

Looking at the maximum precision used by QSopt_ex in obtaining an optimal basis, 51.9% were solved in plain double arithmetic, 74.0% could be solved using at most 128-bit representations, 81.6% could be solved using 192-bit representations, and 98.9% of the problems could be solved using 256-bit representations.

In Fig. 1 we show the empirical distribution of the percentage relative errors in the objective values reported by the original QSopt code. Note that 95% of the problems have relative error of less than 0.0001% in the objective value.

4.2. Orthogonal-array bounds

Orthogonal arrays are used in statistical experiments that call for a fractional factorial design. In such

applications, columns correspond to the factors in the experiment, and the rows specify the levels at which observations are to be made.

Consider two sets of factors F_1 and F_2, with $|F_1| = k_1$, $|F_2| = k_2$, where all factors in F_1 can have s_1 different levels and all factors in F_2 can have s_2 levels. An *orthogonal array* in this case is a matrix M where each row is a (k_1+k_2)-tuple in $\{1,\ldots,s_1\}^{k_1} \times \{1,\ldots,s_2\}^{k_2}$, such that in any submatrix M' of M with t columns, all possible t-tuples that could occur as rows appear equally often. Let n denote the number of rows of M, the orthogonal array for $F_1 \cup F_2$is said to have *strength* t and *size* n.

Sloane and Stufken [21] introduced an LP problem that produces a lower bound on the size of these arrays. They were able to compute bounds for configurations having $s_1 = 2$, $s_2 = 3$, $t = 3$, $k_1 <= 60$ and $k_1+2k_2 \leqslant 70$ using an early implementation of CPLEX (Version 1.2), but found that "Outside this range the coefficients in the linear program get too large."

Given positive integers s_1, s_2, k_1, k_2, and t, the Sloan-Stufken LP problem $SS(s_1, k_1, s_2, k_2, t)$ is

$$\begin{aligned}
\min \quad & \sum_{i=0}^{k_1}\sum_{j=0}^{k_2} x_{i,j} \\
\text{s.t.} \quad & x_{0,0} \geqslant 1, \quad x_{i,j} \geqslant 0 \\
& \quad \text{for } 0 \leqslant i \leqslant k_1, \quad 0 \leqslant j \leqslant k_2, \\
& \sum_{i'=0}^{k_1}\sum_{j'=0}^{k_2} P_{s_1}^{k_1}(i,i') P_{s_2}^{k_2}(j,j') x_{i',j'} \geqslant 0, \\
& \quad \text{for } 0 \leqslant i \leqslant k_1, \quad 0 \leqslant j \leqslant k_2, \\
& \sum_{i'=0}^{k_1}\sum_{j'=0}^{k_2} P_{s_1}^{k_1}(i,i') P_{s_2}^{k_2}(j,j') x_{i',j'} = 0, \\
& \quad \text{for } 1 \leqslant i + j \leqslant t,
\end{aligned}$$

where $P_s^k(a,b) = \sum_{j=0}^{b} (-1)^j (s-1)^{b-j} \binom{a}{j}\binom{k-a}{b-j}$.

With more recent versions of the CPLEX solver it is possible to handle larger instances than reported in the Sloane–Stufken tests, but the class remains challenging. For example, the problem $SS(3, 18, 5, 18, 35)$ is reported as infeasible by the CPLEX 9.1 primal simplex, dual simplex, and barrier solvers, although a feasible solution exists. This is typical of instances in this range and would seem to be a problem inherent in any floating-point LP code. Moreover, for larger instances the coefficients of the constraint matrix simply cannot be represented accurately in double precision.

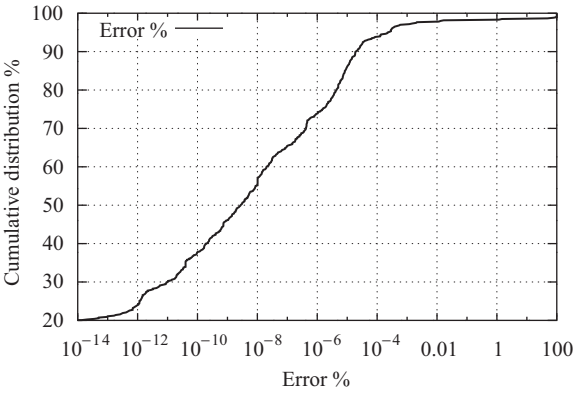

Fig. 1. Relative objective value error distribution.

Table 5
Optimal values of Sloan–Stufken LP problems

#	LP value
1	94327730356522658494464
2	135655450138660858313525821 44
3	116029170155895962680011952696405433460779259002 88

In our experiments we solved a collection of *SS* instances with QSopt_ex. In Table 5 we present the optimal objective values for three large examples,

$SS(10, 18, 10, 19, 18)$,
$SS(10, 36, 10, 37, 18)$,
$SS(15, 56, 15, 57, 28)$.

In all three instances, double precision was sufficient to identify the correct solutions. The running times were 2.53, 3.87, and 77.10 s, respectively. Of course, arrays of the size indicated by the bounds in Table 5 are not of practical use, but the tests indicate the suitability of the QSopt_ex solver for instances with extremely large input data (the test instances have constraint-matrix coefficients with over 150 bits).

4.3. TSP-related tests

An instance of the TSP can be specified by a complete graph with vertex set V, edge set E, and edge weights $(c_e : e \in E)$ giving the cost of travel between

Table 6
Exact subtour bounds

Instance	Time (s)	Subtour bound
rl11849	365	21935527/24
usa13509	387	79405855/4
brd14051	341	21020743/45
d15112	451	375571207/240
d18512	587	92464823/144
pla33810	4528	525643505/8
pla85900	32106	141806385
E1M.0	3.8×10^6	41679386539494215/58810752

each pair of vertices. For a proper subset $S \subseteq V$, let $\delta(S) \subseteq E$ denote the set of edges having one end in S and one end not in S. With this notation, the well known *subtour relaxation* of the TSP is given by the LP problem

$$\begin{aligned} \min \quad & \sum (c_e x_e : e \in E) \\ \text{s.t.} \quad & \sum (x_e : e \in \delta(\{v\})) = 2 \quad \forall\, v \in V, \\ & \sum (x_e : e \in \delta(S)) \geqslant 2 \quad \forall\, \emptyset \neq S \subsetneq V, \\ & 0 \leqslant x_e \leqslant 1 \quad \forall\, e \in E. \end{aligned}$$

Numerous studies report lower bounds obtained by solving this LP relaxation, but these are typically carried out with floating-point approximations of the problem. Using QSopt_ex as a subroutine, we have computed exact subtour solutions for all instances in the TSPLIB [20]. The results for all examples having at least 10,000 cities are given in Table 6; the instance `E1M.0` is a million-city random-Euclidean example studied in [7].

Our implementation relies on Concorde [1] to obtain a floating-point approximation for the subtour LP. We then use our exact LP solver, iterating the cutting-plane and column-generation process in rational arithmetic, until we prove optimality. To detect violated subtour inequalities we have implemented a version of the Padberg and Rinaldi [19] minimum-cut algorithm in rational arithmetic.

It should be noted that most of the time is spent in the routine for pricing the complete edge set in rational arithmetic. Indeed, in the case of the million-city TSP, the operations of solving the current LP relaxation and finding minimum-cuts took under 400 seconds of the 3.8×10^6 total.

Table 7
Exact values of MIPLIB problems

Problem	Value
fiber	20296759/50
fixnet6	3983
gesa2 and gesa2_o	26480487186044893057443711611781/ 10271774522031920000000000
manna81	−13,164
p2756	3124

4.4. Mixed-integer programming

Neumaier and Shcherbina [18] show that, for some seemingly innocent problems with all variables integer, state-of-the-art MIP solvers fail to find optimal solutions.

Using the QSopt_ex library, we created an exact MIP solver aimed at modest-sized instances. The solver consists of a branch-and-cut procedure with exact versions of lifted-cover inequalities [12] and Gomory mixed-integer cuts [10], including the scaling technique of Cornuéjols et al. [4]. Branching is carried out with a version of pseudocost variable selection [17]. We adopt the straightforward approach of using QSopt_ex to solve each LP that is encountered; a more efficient method would attempt to use approximate dual solutions to prune the branch-and-bound search tree when possible.

In Table 7 we report the optimal values for six instances from the MIPLIB 2003 collection. Other problems from MIPLIB can also be solved with this code, but, in general, running times are larger than commercial branch-and-bound implementations by two or three orders of magnitude. The running times for the six instances in Table 7 ranged from 58 s for mann81 to 23 h for gesa2_o.

Acknowledgment

This research project was partly funded by ONR Grant N00014-03-1-0040 and by NSF Grant DMI-0245609.

References

[1] D. Applegate, R.E. Bixby, V. Chvátal, W. Cook, Implementing the Dantzig–Fulkerson–Johnson algorithm for large traveling salesman problems, Math. Prog. 97 (2003) 91–153.

[2] D. Applegate, S. Dash, W. Cook, QSopt, Available at ⟨www.isye.gatech.edu/~wcook/qsopt⟩.
[3] D. Applegate, S. Dash, W. Cook, D. G. Espinoza, QSopt_ex, Available at ⟨www.dii.uchile.cl/~daespino⟩.
[4] G. Cornuéjols, Y. Li, D. Vandenbussche, K-cuts: a variation of Gomory mixed integer cuts from the lp tableau, INFORMS J. Comput. 15 (4) (2003) 385–396.
[5] M. Dhiflaoui, S. Funke, C. Kwappik, K. Mehlhorn, M. Seel, E. Schömer, R. Schulte, D. Weber, Certifying and repairing solutions to large LP's how good are LP-solvers? In: SODA 2003, Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms, 2003, pp. 255–256.
[6] D.G. Espinoza, On linear programming, integer programming and cutting planes, Ph.D. Thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, March 2006.
[7] M.L. Fredman, D.S. Johnson, L.A. McGeoch, G. Ostheimer, Data structures for the traveling salesmen, J. Algorithms 18 (1995) 432–479.
[8] GAMS, World performance libraries, 2006, Available at ⟨www.gamsworld.org/performance/performlib.htm⟩.
[9] B. Gärtner, Exact arithmetic at low cost—a case study in linear programming, Comput. Geom. 13 (1999) 121–139.
[10] R.E. Gomory, An algorithm for the mixed integer problem, Technical Report RM-2597, RAND Corporation, 1960.
[11] T. Granlund, the GNU multiple precision arithmetic library, Available at ⟨http://www.swox.com/gmp/⟩.
[12] Z. Gu, G.L. Nemhauser, M.W.P. Savelsbergh, Lifted cover inequalities for 0-1 integer programs: computation, INFORMS J. Comput. 10 (1998) 427–437.
[13] ILOG CPLEX Division, Incline Village, Nevada, 89451, USA, User's Manual, ILOG CPLEX 10.0, 2006.
[14] C. Jansson, Rigorous lower and upper bounds in linear programming, SIAM J. Options 14 (2004) 914–935.
[15] M. Kiyomi, exlp, an exact LP solver, 2002, Available at ⟨members.jcom.home.ne.jp/masashi777/exlp.html⟩.
[16] T. Koch, The final NETLIB-LP results, Oper. Res. Lett. 32 (2003) 138–142.
[17] J.T. Linderoth, M.W.P. Savelsbergh, A computational study of search strategies for mixed integer programming, INFORMS J. Comput. 11 (1999) 173–187.
[18] A. Neumaier, O. Shcherbina, Safe bounds in linear and mixed-integer linear programming, Math. Prog. 99 (2004) 283–296.
[19] M.W. Padberg, G. Rinaldi, An efficient algorithm for the minimum capacity cut problem, Math. Prog. 47 (1990) 19–36.
[20] G. Reinelt, TSPLIB95, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg, 1995.
[21] N.J.A. Sloane, J. Stufken, A linear programming bound for orthogonal arrays with mixed levels, J. Stat. Plann. Inference 56 (1996) 295–306.
[22] D.H. Wiedemann, Solving sparse linear equations over finite fields, IEEE Trans. Inf. Theory 32 (1986) 54–62.