

**An asynchronous parallel revised
simplex algorithm**

J.A.J. Hall K.I.M. McKinnon

MS 95-050

Sept 1995

Department of Mathematics and Statistics

University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ

Tel. (33) 131 650 5075 E-Mail : jajhall@maths.ed.ac.uk, ken@maths.ed.ac.uk

An asynchronous parallel revised simplex algorithm

J. A. J. Hall K. I. M. McKinnon

2nd September 1995

Abstract

This paper describes a asynchronous variant of the revised simplex method which is suitable for parallel implementation on MIMD computers with fast inter-processor communication. The method overlaps simplex iterations on different processors. Candidates to enter the basis are tentatively selected using reduced costs which may be out of date. Later the up-to-date reduced costs of the tentative candidates are calculated and candidates are either discarded or accepted to enter the basis. The implementation of this algorithm on a Cray T3D is described and results demonstrating significant speed-up are presented.

1 Introduction

Linear programming (LP) is a widely applicable technique both in its own right and as a sub-problem in the solution of other optimization problems. The revised simplex method and the barrier method are the two efficient methods for general LP problems on serial machines. There have been successful parallel implementations of the barrier method but as yet little progress has been reported on parallel methods based on the revised simplex algorithm. In contexts where families of related LP problems have to be solved, such as in integer programming and decomposition methods, the revised simplex method is the more efficient method, so there is strong motivation to devise a parallel version of this method.

The major computational steps in the revised simplex method are described later in this section and general approaches to exploiting parallelism are discussed. The particular approach to exploiting parallelism which is considered in this paper is to overlap simplex iterations performed on

a number of *iteration processors*, with an additional *invert processor* devoted to calculating a factored inverse of simplex basis matrices.

Candidates for variables to enter the basis are tentatively selected using the most recently available reduced costs, and the true reduced costs for these columns are calculated cheaply later before finally deciding whether or not they will enter the basis. The effectiveness of the method relies on there being some persistence in the values of the reduced costs in the course of a small number of iterations. The algorithm is a variant of the simplex method and as such follows a single path on the surface of the feasible region. This requires the coordination of the basis change decisions among all the processors.

A parallel algorithm and a description of its implementation on a Cray T3D are given in Section 2. Results demonstrating significant speed-up for problems from the Netlib test set [3] are presented in Section 2.5 and conclusions are discussed in Section 3.

1.1 Background

The two main variants of the simplex method are the standard simplex method and the revised simplex method. Although early versions of the revised simplex method used an explicit form of the inverse, this was quickly replaced by methods based on a factored form of the inverse. Most important LP problems are large (some with tens of thousands of variables and constraints) and sparse (the coefficient matrix has about six non-zeros per column). An important point to note is that for large sparse problems the standard simplex and the explicit inverse form of the revised simplex are completely uncompetitive in speed compared with the revised simplex method when a factored form of the inverse is used. They are also much less stable numerically.

There have been several studies such as those by Luo *et. al* in [9] and Stunkel in [11] which have implemented either the standard form of the simplex or the revised simplex with the inverse of the basis matrix stored as a full matrix. Both methods parallelise well but, as noted above, are so bad for large sparse problems, that the results are a lot slower than a good serial implementation.

A simple parallel implementation of the revised simplex method using a sparse LU decomposition is described by Shu and Wu in [10]. However they only parallelise the PRICE operation and report little or no speed-up over their serial implementation.

BTRAN: Form $\boldsymbol{\pi}^T = \mathbf{c}_B^T B^{-1}$.
 PRICE: Calculate the *reduced costs* $\hat{\mathbf{c}}_N^T = \mathbf{c}_N^T - \boldsymbol{\pi}^T N$.
 CHUZC: Scan $\hat{\mathbf{c}}_N$ for the best candidate q to enter the basis.
 If no such candidate exists then exit (optimality).
 FTRAN: Form $\hat{\mathbf{a}}_q = B^{-1} \mathbf{a}_q$, where $\mathbf{a}_q$ is column q of A .
 CHUZR: Scan the ratios $\frac{\hat{b}_i}{\hat{a}_{iq}}$ for a good candidate to leave the basis,
 where $\hat{\mathbf{b}} = B^{-1} \mathbf{b}$ (ratio test).
 If {growth in factors} then
 INVERT: Find a factored inverse of B .
 else
 UPDATE: Update the inverse of B corresponding to the basis change.
 endif

Figure 1: The revised simplex method

1.2 The revised simplex method

A linear programming problem has the form

$$\begin{aligned}
 &\text{maximize} && f = \mathbf{c}^T \mathbf{x} \\
 &\text{subject to} && \mathbf{x} \geq \mathbf{0} \\
 &&& A\mathbf{x} = \mathbf{b} \\
 &\text{where} && \mathbf{x} \in \mathbb{R}^n \quad \text{and} \quad \mathbf{b} \in \mathbb{R}^m.
 \end{aligned} \tag{1}$$

At any stage in the simplex method the variables are partitioned into two sets, basic variables $\mathbf{x}_B$ and nonbasic variables $\mathbf{x}_N$. The set of basic variables is referred to as the basis. If the problem is partitioned correspondingly then the objective function is $f = \mathbf{c}_B^T \mathbf{x}_B + \mathbf{c}_N^T \mathbf{x}_N$, the constraints are $B\mathbf{x}_B + N\mathbf{x}_N = \mathbf{b}$ and the basis matrix B is nonsingular. Each basic variable is identified with a particular row of the constraint matrix A , and each nonbasic variable is identified with a particular column of the matrix N . The major computational steps of the revised simplex method are illustrated in Figure 1.

At the beginning of an iteration of the revised simplex method it is assumed that a factored inverse of the basis matrix B is available, i.e. elementary row or column matrices $M_1, M_2, \dots, M_r$ are known such that $B^{-1} = M_1 M_2 \dots M_r$. The first operation is the calculation of the dual variables $\boldsymbol{\pi}^T = \mathbf{c}_B^T B^{-1}$ by passing backwards through the factors of B^{-1} , an operation known as BTRAN. This is followed by the PRICE operation, which is a sparse matrix-vector product which yields the reduced costs of the nonbasic

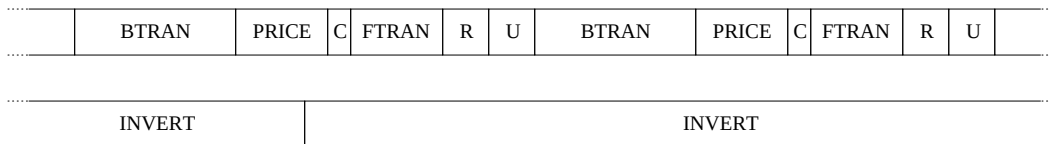
variables. These reduced costs are scanned in the operation known as CHUZC to find a good candidate q , say, to enter the basis. This procedure was traditionally based on the *Dantzig* criterion of selecting the candidate with the largest reduced cost. For many problems it is more efficient to use an exact or approximate steepest edge criteria, as described by Harris in [8] and by Goldfarb and Reid in [6]. However a discussion of these techniques in the parallel context is beyond the scope of this paper.

In order to determine the basic variable which would be replaced by the candidate to enter the basis it is necessary to calculate the column of the standard simplex tableau corresponding to this candidate. This *pivotal column* $\hat{\mathbf{a}}_q = B^{-1}\mathbf{a}_q$, where $\mathbf{a}_q$ is column q of the constraint matrix A , is formed by passing forward through the factors of B^{-1} , an operation known as FTRAN. The row corresponding to the leaving variable is determined by the CHUZR operation which scans the ratios $\hat{b}_i/\hat{a}_{iq}$, where $\hat{\mathbf{b}} = B^{-1}\mathbf{b}$ is the vector of current values of the basic variables. Traditionally the leaving variable is that corresponding to the smallest non-negative ratio since this ensures that no variable exceeds its bounds. However it is preferable to use a selection criterion based on finding the most numerically advantageous candidate from those which lead to bound violations which are sufficiently small. Such techniques are described by Harris in [8] and by Gill *et al* in [5] and are of particular advantage in the parallel revised simplex method algorithms discussed in this paper. This is because in the parallel algorithm there are always factors in the current invert which date from before the most recent reinversion, and this leads to a reduction in numerical stability.

Once a basis change has occurred, the inverse of the current basis matrix is updated unless it is either more efficient or necessary for numerical stability to find a new factored inverse. Various methods are possible for the UPDATE operation. Let us denote the basis invert produced by INVERT by B_0 and the inverse k basis changes later by B_k . The simplest form of update is the product form update [2], which represents B_k^{-1} as

$$S_{k-1}^{-1} \dots S_0^{-1} B_0^{-1}. \quad (2)$$

The matrix S_j is an elementary column matrix whose only non unit column is simply derived from the vector $\hat{\mathbf{a}}_q$ for the variable to enter the basis in that iteration. The Bartels-Golub and Forrest-Tomlin updates modify the factors of B_0 in order to reduce the size of the factored inverse of B_k . Update procedures based on the use of a Schur complement are described by Gill *et al* in [4].



C: CHUZO R: CHUZR U: UPDATE

Figure 2: Overlapping INVERT with simplex iterations

1.3 General approaches to parallelising the revised simplex method

With the exception of INVERT, the computational steps of the revised simplex method, as presented in Figure 1, cannot be performed independently. The immediate scope for parallelism is, therefore, to dedicate one processor to performing the INVERT operations and another to performing simplex iterations. This is illustrated by the Gantt chart in Figure 2.

Note that this approach also results in an increased frequency of INVERT. This reduces the size of the factors of the inverse, leading to a reduction in the time required for BTRAN and FTRAN. One consequence of this approach is that the basis matrix which is inverted is no longer current when INVERT is complete since basis changes have occurred as a result of simplex iterations performed on the other processor. Thus, before it can be used to perform BTRAN and FTRAN, it must be brought up-to-date with respect to these basis changes. This issue is discussed by Hall and McKinnon in [7] where, due to the cost of bringing the inverse up-to-date, it is concluded that the use of Bartels-Golub or Forrest-Tomlin updates is inappropriate. However, when product form or Schur complement update procedures are used, the inverse can be brought up-to-date in an efficient manner.

The efficiency of simply overlapping INVERT with simplex iterations of this approach is limited due to the fact that INVERT usually accounts for less than 10% of execution time for a serial LP code. Set against the saving is the overhead due to communicating the factored inverse and bringing it up-to-date.

Any parallel algorithm for the revised simplex method must either exploit parallelism within computational steps of the revised simplex method or overlap computational steps. For sparse problems, although it is easy to parallelise PRICE, CHUZO and CHUZR, the parallelism which may be exploited within BTRAN and FTRAN is limited, very fine grained and hard to achieve. For this reason the algorithm described in Section 2 exploits

parallelism by using several processors to overlap the computational steps required to perform simplex iterations, and using one further processor to perform inverts.

2 An asynchronous parallel algorithm

The path followed by the simplex method is determined by the sequence of pairs of variables entering and leaving the basis. Except in degenerate or near degenerate iterations there is no choice in the variable to leave the basis (in CHUZR). There is however normally considerable flexibility in the choice of variable to enter the basis (in CHUZC) and indeed, as mentioned earlier, different schemes have been proposed for this. Provided the variable has a negative reduced cost, a valid simplex iteration will occur. Once the FTRAN is complete the reduced cost for that variable, q say, can be calculated from $c_q - \mathbf{c}_B^T \hat{\mathbf{a}}_q$, which is a quick operation. One possible strategy is therefore to make a tentative choice of the column to enter the basis, perform the FTRAN operation and then check the reduced cost. If the column has a negative reduced cost then the iteration can continue, and if not the column has to be rejected and the work done in performing the FTRAN is wasted.

The parallel algorithm proposed in this section exploits parallelism by overlapping computational steps which are executed sequentially rather than by exploiting parallelism within any individual computational step. At any stage where a processor is idle it is given a tentative candidate to enter the basis using the most up-to-date reduced costs which are available. Whenever a basis change occurs, BTRAN and PRICE are performed in order to obtain a new set of reduced costs. This produces the most up-to-date reduced costs possible. However, for some classes of problem it may be better to calculate the reduced costs less frequently and accept reduced costs which are even more out-of-date than is inevitable in an asynchronous algorithm.

2.1 Overlapping simplex iterations

One invert processor performs INVERT and p iteration processors perform the computational steps required to determine iterations of the revised simplex method. When an iteration processor receives a tentative candidate to enter the basis it performs FTRAN and then brings the pivotal column (and the values of the basic variables) up-to-date with respect to basis changes which have been received from other processors during the course of FTRAN. The basis change is determined by CHUZR and then the basis change and the pivotal column are broadcast to the other iteration processors so that they



C: CHUZYC R: CHUZR U: UPDATE A: APPLY

Figure 3: Overlapping simplex iterations

can bring themselves up-to-date. The basis change is then sent to the invert processor where it will be incorporated into the next basis matrix to be inverted. The processor which has just completed CHUZR now calculates reduced costs with respect to the new basis by performing BTRAN and PRICE.

This regular overlapping of iterations is illustrated in Figure 3 for the case when $p = 4$. Iteration processor i determines the basis change associated with iterations $i, p+i, 2p+i, \dots$. As a result, the reduced cost of a chosen column is $p - 1$ basis changes out-of-date when that column enters the basis.

2.2 Limit on speedup by overlapping iterations

There are a number of factors which limit the speedup which can be achieved by overlapping iterations as illustrated in Figure 3. These are duplicated work, wasted work, communication time and an increase in the number of iterations with increasing number of processors.

Let the average time required to perform one of each of the simplex operations BTRAN, PRICE, CHUZYC, FTRAN, CHUZR and UPDATE be denoted by S , the time for FTRAN by F , the time for CHUZR by R , and the time taken to apply a basis change received from another processor, APPLY, by A . The number of simplex iterations required to solve a problem depends on the number of processors p and will be denoted by N_p . Since the reduced cost of a column is not up-to-date when the variable enters the basis, this column may well not be the best candidate to enter the basis in the particular iteration, so using of out-of-date costs is expected to increase the number of iterations required to solve the LP problem. It is also possible that the column no longer has a negative reduced cost, in which case it must

be discarded, so that FTRAN is wasted. The number of wasted FTRANs depends on p and will be denoted by n_p . The more out of date the costs are the more wasted FTRANs will occur. The variation of N_p and n_p with p is highly problem-dependent.

Neglecting communication, and assuming all operations can be overlapped, the time taken to perform these simplex iterations using p iteration processors is

$$\hat{T}(p) = \frac{N_p}{p}(S + (p-1)A) + \frac{n_p}{p}F$$

So long as N_p and n_p increase less than linearly with p , this tends to the limit A as $p \rightarrow \infty$. A is the time of an APPLY operation, and this is the only significant duplicated work in the algorithm. Note that the APPLY operation is a very small proportion of the work of an iteration.

Since CHUZR cannot be started on one processor until the previous CHUZR has been performed on another processor and the resulting basis change has been communicated, received and applied, the time for these operations limits the maximum rate at which simplex iterations can be performed. The makespan of the algorithm is therefore given by

$$T(p) = \max(\hat{T}(p), N_p(A + R)),$$

and, as the number of processors is increased, $N_p(A + R)$ becomes the limiting time to solve the LP problem.

2.3 An asynchronous parallel algorithm

The algorithm presented below in Figures 4-7 allows simplex iterations to be overlapped using p iteration processors. The algorithm ensures that candidates are chosen from the most up-to-date reduced costs available and maintains processor activity in the event of candidates being found to be unattractive.

Clearly no two processors should simultaneously be forming the standard simplex tableau column for the same candidate variable. Initially it is possible to distribute the p most attractive candidates over all the iteration processors. However, as soon as one basis change has been determined, it is no longer straightforward to decide which candidate to choose according to the new set of reduced costs. It is quite likely that the most attractive candidate is currently being processed by one of the other processors. What is required is some mechanism to filter out the attractive candidates which have been chosen by other processors but which have not yet entered the

```

Repeat
  Repeat
    Apply basis change
  until {All basis changes received have been applied}
  INVERT
  Send new factored inverse
until {Simplex algorithm terminates}

```

Figure 4: Invert processor

basis. In the algorithm presented below, a set of the most attractive columns not currently chosen as candidates to enter the basis is determined after each basis change. Not only does this provide the next chosen column for the processor on which the corresponding reduced costs were calculated but the most up-to-date set provides a pool of immediately-available attractive candidates in the event of a candidate chosen previously proving to be unattractive.

Once a basis change has been determined, the indices of the variables entering and leaving the basis, together with the corresponding pivotal column, are communicated to all other iteration processors. Since the time required to perform FTRAN for different candidates can vary significantly, it is quite possible for two processors to have received and applied all basis changes which have been determined. A management scheme is necessary to determine which of these processors performs CHUZR and which waits to receive the basis change from the other.

Although it is logically possible to manage column selection and basis changes by passing short fast messages between the iteration processors, for ease of implementation it was decided to use two additional processors in order to perform these management tasks. These processors are largely idle and their role is expected to be eliminated in future parallel implementations.

2.4 Implementation

A parallel revised simplex code using the algorithm described above has been written for the Cray T3D based at Edinburgh University. This machine has a very high ratio of communication to computation speed. The SHMEM suite of inter-processor communication routines allow a bandwidth of up to 120MB/s with negligible latency. The processing elements are Dec Alpha chips with 30Mflop peak performance. However, this computational performance will not be approached due to the indirect addressing of arrays

```

 $k_i = 0$ 
BTRAN
PRICE
Let  $q$  be the  $i^{\text{th}}$  most attractive candidate
Repeat
  If {Received a new factored inverse} Install new factored inverse
  FTRAN
1 Continue
  Repeat
    update  $\hat{\mathbf{a}}_q$ 
    update  $\hat{c}_q$ 
    update  $\hat{\mathbf{b}}$ 
    update eta file
    apply basis change
     $k_i := k_i + 1$ 
  until {All basis changes received have been applied}
  If  $\{\hat{c}_q > 0\}$  then
    Send a message that the candidate is unattractive
  else
    Send an offer to perform CHUZR
    Wait for a reply to offer
    If {Offer accepted} then
      CHUZR
      Send basis change and pivotal column to other workers
      Send basis change to master and column selection manager
      UPDATE
       $k_i := k_i + 1$ 
      BTRAN
      PRICE
      Choose a set of the most attractive candidates
      Send the most attractive candidates
    else
      Wait to receive next basis change
      goto 1
    endif
  endif
  Wait for a new candidate column
until {Simplex algorithm terminates}

```

Figure 5: Iteration processor i

```

 $k_c = 0$ 
Repeat
  If {Received basis change} then
    apply basis change
  else if {Received from processor  $i$  a set of candidates corresponding
to
    basis  $k_i$ } then
    If  $\{k_i > k_c\}$  then
      Filter out the candidates already sent to other processors
       $k_c = k_i$ 
    endif
    Send to processor  $i$  a candidate to enter the basis and mark the
    candidate as sent
  else if {Received from processor  $i$  a message that its current
candidate is
    now unattractive} then
    Send to processor  $i$  a candidate to enter the basis and mark the
    candidate as sent
  endif
until {Simplex algorithm terminates}

```

Figure 6: Column selection manager

```

 $k_r = 0$ 
Repeat
  If {Received from processor  $i$  an offer to perform CHUZR corresponding
to basis  $k_i$ } then
    If  $\{k_i > k_r\}$  then
      Send an acceptance of the offer
       $k_r = k_i$ 
    else
      Send a refusal of the offer
    endif
  endif
until {Simplex algorithm terminates}

```

Figure 7: Row selection manager

Problem	ERGOL			OSL		
	CPU (s)	Simplex iterations	Iteration rate	CPU (s)	Simplex iterations	Iteration rate
SHELL	3.25	774	4.2×10^{-3}	3.79	751	5.0×10^{-3}
SCTAP3	12.3	1630	7.5×10^{-3}	21.7	1677	1.3×10^{-2}
25FV47	95.8	4922	1.9×10^{-2}	62.7	4164	1.5×10^{-2}
GREENBEB	358.	11636	3.1×10^{-2}	424.	11869	3.6×10^{-2}

Table 1: Comparison of ERGOL and OSL

which is associated with large sparse problems.

The parallel code was developed using modules of our serial code ERGOL. The efficiency of this code may be judged from the comparisons with OSL [1] given in Table 1. The CPU times on a SUN SPARCstation 5 are given for the four Netlib [3] problems for which numerical results with the parallel code are discussed in Section 2.5.

A prototype code was developed using PVM for message-passing but the communication overhead, in particular the high latency of short messages, led to poor practical results. The result of replacing PVM by SHMEM is that the communication overhead can be largely ignored.

The inherent numerical instability of the product form update is magnified in the context of the parallel algorithm above. Iteration processors may detect excessive growth due to choosing a small pivot value, and the invert processor may, ultimately, detect a singular basis. As a result, a facility was required in order to reset the parallel LP code from a new factored inverse of a particular basis. A description of this procedure is beyond the scope of this paper.

2.5 Numerical results

Numerical results for four problems from the Netlib [3] test set are presented in Tables 2-5. The first column of each table of results gives the number of iteration processors, with zero corresponding to the serial code running on one processor. The number of simplex iterations required to solve the problem is given in the second column. Column three gives the total number of candidates which prove to be unattractive when their pivotal column is brought up-to-date. The final column gives the total CPU time in seconds to solve the problem, starting from a logical basis.

The difference in the number of simplex iterations when the serial code is used and the case where just one processor performs simplex iterations is explained by the fact that different basis matrices are inverted, leading to

Iteration processors	Simplex iterations	Unattractive candidates	CPU (s)
0	774	-	3.0
1	759	0	3.6
2	772	198	2.1
3	766	312	1.5
4	742	341	1.2
5	762	469	0.93
6	803	421	0.85
8	814	779	0.79
10	761	601	0.66
12	783	977	0.62

Table 2: SHELL: 537 rows 1775 columns 4900 nonzeros

Iteration processors	Simplex iterations	Unattractive candidates	CPU (s)
0	1681	-	15.8
1	1589	0	15.0
2	1949	520	10.4
3	2021	1005	7.7
4	2119	1387	6.6
5	2299	1410	6.4
6	2390	1876	6.0
8	2333	2418	5.0
10	2453	2843	4.9
12	2363	2771	4.5

Table 3: SCTAP3: 1481 rows 2480 columns 10734 nonzeros

different rounding and hence different basis changes at degenerate vertices.

For the first two problems there is no gain in efficiency by performing INVERT in parallel with just one simplex iteration processor. However there is a gain in efficiency for 25FV47 and GREENBEB. This is partly because INVERTs are relatively more expensive for these problems but mainly due to the significant reduction in the average time for FTRAN and BTRAN when new factored inverses are available more frequently.

For SHELL, the number of iterations does not increase with the number of iteration processors so a significant speed-up over the time required for the serial code is achieved.

For SCTAP3 there is some increase in the number of simplex iterations

Iteration processors	Simplex iterations	Unattractive candidates	CPU (s)
0	5003	-	107
1	5263	0	74
2	6561	3247	62
3	7111	5542	51
4	8171	8825	52
5	8245	10710	46
6	7678	12082	40
8	9279	16760	44

Table 4: 25FV47: 822 rows 1571 columns 11127 nonzeros

Iteration processors	Simplex iterations	Unattractive candidates	CPU (s)
0	11821	-	384
1	11914	0	319
2	14679	5852	231
3	15883	10560	193
4	16848	14966	171
5	15444	16898	137
6	17692	21057	144
8	20556	28532	155
10	18077	29873	119

Table 5: GREENBEB: 2383 rows 5405 columns 31499 nonzeros

so the efficiency of using several iteration processors is not so marked as with SHELL.

The practical performance of the parallel solver when applied to 25FV47 is less good than with the first two problems. The number of iterations and unattractive candidates increases significantly, limiting the speed-up that can be achieved.

GREENBEB is a medium sized problem and, although the candidate persistence is poor, FTRAN is relatively inexpensive so significant speed-up is still achieved.

A typical Gantt chart of actual processor activity when solving LP problems is illustrated in Figure 8. Processor activity is almost total, although some of this will be spent calculating pivotal columns for candidates which prove to be unattractive. Light grey boxes correspond to time associated with communication and, with the exception of the time spent

in a similar manner, allowing the expected total number of simplex iterations to be reduced.

Although most of the communication overhead has been minimized by the use of SHMEM routines, the cost of broadcasting the new factored inverse is still significant. This factored inverse may only be applied a few times on each processor and it may prove more efficient to communicate the inverse to just one or two processors which would then communicate partially FTRANned columns and complete BTRAN for columns whose BTRAN operation has been started elsewhere.

These algorithmic developments are currently being considered and are expected to form the basis for future work.

References

- [1] *Optimization Subroutine Library, guide and reference, release 2.*
- [2] G. B. Dantzig and W. Orchard-Hays. The product form for the inverse in the simplex method. *Math. Comp.* , 8:64–67, 1954.
- [3] D. M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 13:10–12, 1985.
- [4] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Sparse matrix methods in optimization. *SIAM J. Sci. Stat. Comput.* , 5:562–589, 1984.
- [5] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linear and nonlinear programming. Technical Report SOL 88-4, Systems Optimization Laboratory, Stanford University, 1990.
- [6] D. Goldfarb and J. K. Reid. A practical steepest-edge simplex algorithm. *Math. Prog.*, 12:361–371, 1977.
- [7] J. A. J. Hall and K. I. M. McKinnon. Update procedures for the parallel revised simplex method. Technical Report MSR 92-13, Department of Mathematics and Statistics, University of Edinburgh, 1992.
- [8] P. M. J. Harris. Pivot selection methods of the Devex LP code. *Math. Prog.*, 5:1–28, 1973.

- [9] J. Luo, A. N. M. Hulsbosch, and G. L. Reijns. An MIMD work-station for large LP problems. *Parallel Processing and Applications*, pages 159–169, 1988.
- [10] W. Shu and M. Wu. Sparse implementation of revised simplex algorithms on parallel computers. In *Proceedings of 6th SIAM Conference on Parallel Processing for Scientific Computing*, pages 501–509, 1993.
- [11] C. B. Stunkel. Linear optimization via message-based parallel processing. *ICPP*, 3:264–271, 1988.