Steepest Edge as Applied to the Standard Simplex Method

Gavriel Yarmish, Brooklyn College
City University of New York
(yarmish@sci.brooklyn.cuny.edu)

Abstract

In this paper we discuss results and advantages of using steepest edge column choice rules and their derivatives. We show empirically, when we utilize the steepest edge column choice rule for the tableau method, that the density crossover point at which the tableau method is more efficient than the revised method drops to 5%. This is much lower than the 72% we have seen when the tableau method used the classical column choice rule. This is because the revised method loses much of its gain in terms of lower iteration count due to the extra computation necessary in applying steepest edge rules. This can also be seen via a theoretical analysis.

1. Introduction

Dantzig's simplex algorithm for linear programming has two major variants: the original, or *standard tableau* method, and the *revised* method. Today, virtually all serious implementations are based on the revised method because it is much faster for sparse LPs, which are most common. However, the standard method has advantages as well. First, the standard method is effective for dense problems. While dense problems are uncommon in general, they occur frequently in some important applications such as wavelet decomposition, digital filter design, text categorization, image processing and relaxations of scheduling problems [6, 9, 31].

Second, the standard method can be easily and effectively extended to a coarse grained, distributed algorithm [32]. There have been a number of parallel implementations for the revised method but they have had limited success. Bürger et al proposes a distributed peer to peer algorithm to solve degenerate linear programs [5], Ploskas et al [25] presents a parallel shared memory implementation of the revised method. Lalami et al [19, 20] discusses a parallel implementation via using CPU-GPU processors. There have been many other attempts. For a review of various parallel implementations of the simplex algorithm see Hall [17]. Bixby and Martin [4] attempted a distributed version of the revised method utilizing the steepest edge rule. They report that there is little hope achieving good parallel performance with a distributed-memory model. They also tried with shared-memory up to a maximum of four processors with mixed results. Work has been done that takes advantage of problems with special structures. Hall and Hangfu have written revised simplex code that takes advantage of hyper-sparsity and more recently Hall has written a distributed version of the revised method that takes advantage of problems with a block-structure [15, 16]. At this point there are no truly scalable distributed versions of the revised simplex method for general linear programming problems.

Finally, nonstandard column choice rules, in particular the steepest edge rule, are efficiently implemented in the standard tableau simplex method. There has been much discussion of utilizing alternate column choice rules for the revised method [11, 14, 21, 24, 27–29] including ways to make it more efficient. We show that when utilizing the tableau method the advantages are much more pronounced and we discuss results and advantages of using steepest edge column choice rules and their derivatives.

We show empirically, when we utilize the steepest edge column choice rule for the tableau method, that the density crossover point at which the tableau method is more efficient than the revised method drops to 5%. This is much lower than the 72% we have seen when the tableau method used the classical column choice rule. This is because the revised method loses much of its gain in terms of lower iteration count due to the extra computation necessary in applying steepest edge rules. This can also be seen via a theoretical analysis.

We also provide a theoretical analysis of the extra computational cost of using steepest edge methods tableau vs. revised and show how it is advantageous to the tableau method but not so much so for the revised method.

The steepest edge rule amongst other alternate column choice rules has been known in theory as far back as the simplex method itself. Tests to empirically compare the number of iterations on a number of different column choice rules were performed by Kuhn, Quant, Cutler and Wolf in the early ‘60s [13, 30]. Both the steepest edge rule and the greatest change rule were both empirically shown to require fewer iterations than the classical Dantzig rule. On the other hand, the cost per iteration is higher when using these rules.

Our empirical study did not include the revised method running the steepest edge update methods due to the fact that the implementation of the revised method we had access to did not have an option for the steepest edge. Only some revised implementations include it due to the difficulties described in this paper.

Nevertheless, we do include empirical evidence based on Eigen [10], that computational gains for the revised method of lowering the number of iterations is almost wiped out and in some cases is negative due to the extra computational per iteration.

In a previous paper [32] we showed that, when using the classical column choice rule, the density crossover point when using distributed parallel processors for a 1,000 x 5,000 size problem can be lowered to less than 10% density down from ~72% for a single processor.

We thus have the following table when comparing single processor revised method with the classical column choice rule to the tableau method when the tableau method uses

- the classical column choice rule and a single processor,
- distributed processors with the classical column choice rule and when it uses the
- steepest edge column choice rule with a single processor:

Density crossover point	Single processor	Optimal distributed processors
Classical CCR	~72%	<10%
Steepest edge update CCR	5%	-

Table 1: Density crossover points for classical vs. steepest edge column choice rules

It would be interesting to see how far the density crossover point would decrease when steepest edge column choice rules are used together with parallel processors.

Section 2 is a brief review of the simplex algorithm, it reviews both tableau and revised method and includes a cost comparison between them. Section 3 reviews results and advantages of using a distributed tableau method. Sections 4 and 5 describe a number of alternative column choice rules and explain the reason that they are computationally difficult for the revised method. Section 6 first reviews the steepest edge update rule for the revised method. It then shows a per-iteration cost analysis; classic vs. steepest edge rules for tableau and revised methods. Section 8 shows experimental results based on Eigen [10] that compares the classical and steepest edge rules, both the exact update rule of Forrest and Goldfarb [11] and the Devex approximations of Harris [18]. This comparison is for the revised method only. Section 9 has the results of an experimental study that shows the density crossover point of the tableau vs. the revised method. This crossover point is given both when the tableau method uses the classical column choice rule and when it uses the steepest edge column choice rule. Finally, section 10 is a summary.

2. Review

2.1 The standard Simplex Method

In this section we briefly outline the simplex algorithm and assume that the reader has some familiarity with it.

We consider linear programs in the general form:

$$\begin{aligned} &\underset{x}{Max}\, z = cx \\ &b^l \le Ax \le b^u \\ &l_j \le x_j \le u_j \ \text{for}\ j = 1,...,n \end{aligned} \tag{1}$$

Or with $y = Ax$ we have:

$$\begin{aligned} &\underset{x}{Maximize}\ z = \sum_{j=1}^{n} c_j x_j \\ &Subject\ to\ \mathrm{y}_i = \sum_{j=1}^{n} a_{ij} x_j \ \ (i = 1,2,...,m) \\ &\qquad l_j \le x_j \le u_j \ \text{for}\ j = 1,...,n; \quad b_i^l \le y_i \le b_i^u \ \text{for}\ i = 1,...,m \end{aligned} \tag{2}$$

$A = \{a_{ij}\}$ is a given m x n matrix, x is an n-vector of decision variables x_j, each with given lower bound l_j and upper bound u_j. The m-vectors b^l and b^u are given data that define constraints. The lower bound, l_j, may take on the value -∞ and the upper bound, u_j, may take on the value +∞. Similarly, some or all of the components of b^l may be -∞, and some or all of b^u may be +∞.

Equation (2) together with an assignment of values to the non-basic variables x is a variant of the *dictionary* representation of Strum and Chvátal [7].

2.2 The Revised Simplex Method

In the standard simplex method, most of the effort in moving from one dictionary, (2), to the next comes from calculating the new a_{ij} and c_j coefficients. In general, most of these coefficients change for each new dictionary. This is particularly onerous if the number of columns, n, is relatively large compared to the number of rows, m. Moreover, sparsity is lost. That is, if most of the data elements are zero in the original dictionary, they fill in very quickly with non-zero values in a few iterations of the standard method. Particularly frustrating is that only a small part of each dictionary is used or even looked at!

To perform an iteration of the simplex method, we require only the following from the dictionary (2) (assuming we are using the classical column choice rule):

1) The objective coefficients c_j j=1,...,n,
2) The constraint coefficients, a_{is} i = 1,...,m, for the pivot column, s, and
3) The current values of the basic variables, y_i.

Item 1) is used to determine the pivot column, and Items 2) and 3) are used to determine the pivot row. In summary, we only use two columns and one row from all the data in the dictionary.

By the early 1950's, George Dantzig and William Orchard-Hays [8] realized that these three elements could be derived from one, fixed, original dictionary together with a changing, *auxiliary data structure* that requires less work to update than the work required to change dictionaries. For most linear programs found in practice, it is more efficient to represent the current dictionary implicitly in terms of the original system and the auxiliary data structure rather than explicitly updating the form (2). Such an approach is called a *revised simplex method* and is more efficient for linear programs that are sparse (low density) and have high *aspect ratio* (n/m).

To explain more about this, it is convenient to recast (2). We rename y_i as $-x_{n+i}$ for i = 1,...,m and reconfigure (2) in matrix form as:

$$\begin{aligned} &\underset{x}{Maximize}\ z = CX \\ &Subject\ to: AX = 0 \\ &L \le X \le U \end{aligned} \qquad (3)$$

where $X = [x_1, x_2, ..., x_n, x_{n+1}, ..., x_{n+m}]$, $C = [c_1, c_2, ..., c_n, 0, ..., 0]$, $N = \{a_{ij}\}$, $A = [N \mid I]$, $L=[l_1,...,l_n, -b_1^l, ...,-b_m^l]$, *and* $U=[u_1,...,u_n, -u_1^l,...,-b_m^l]$.

In this notation, each different directory (2) corresponds to a different basis, B, of A. By adding appropriate multiples of the constraints $AX=0$, to the objective we also maintain zero coefficients for the basic variables in the successive C vectors. These operations can be expressed directly in matrix terms. For example, if we are going to pivot in column s (making the non-basic variable x_s basic) and replace the basic variable corresponding to row r, we premultiply A by P where:

$$P = \begin{pmatrix} 1 & 0 & \cdots & \frac{-a_{1s}}{a_{rs}} & \cdots & 0 \\ 0 & 1 & \cdots & \frac{-a_{2s}}{a_{rs}} & \cdots & 0 \\ & & \ddots & \vdots & & \\ & & & \frac{1}{a_{rs}} & & \\ & & & \vdots & \ddots & \\ 0 & 0 & \cdots & \frac{-a_{ms}}{a_{rs}} & \cdots & 1 \end{pmatrix}$$

Modulo some renumbering of equations and variables, the matrix A is updated iteration by iteration by premultiplying by P matrices. So after some number of iterations, k, the new constraint matrix A' can be given in terms of the original one, A, by $A' = [P^k P^{k-1} \bullet\bullet\bullet P^1]A$. Again within numbering of rows and columns $B^{-1} = P^k P^{k-1} \bullet\bullet\bullet P^1$ is the inverse of the current basis, B.

B^{-1} suffices to obtain from the original matrix *A* all we need at an arbitrary iteration, *k*. So this is our first example of an auxiliary structure. This is called the revised simplex method using the *explicit inverse.*
Clearly, we can represent the basis inverse as a product of the individual P matrices (actually you need only save the column with non-trivial entries and its index) separately as another auxiliary structure. This is called the *product form of the inverse.*
More common today is the *LU decomposition* of B (see, for example, [23], Sections 7.6.1 and A.5]); The LU decomposition offers better numerical stability. Heuristics are used for (i) accomplishing the initial LU decomposition for (ii) the updating of the decomposition, and (iii) determining the frequency of updating. They seek an optimal tradeoff between numerical stability and the maintenance of sparsity corresponding to that of the original matrix *B.* [1, 26]. In this context Step 3, "pivot," corresponds to the updating of the LU decomposition, and its periodic (usually at most every 100 iterations) reinitialization or *refactorization.*

2.3 Cost comparison of the tableau and revised methods for classical column choice rule

With ideal computation, the revised and standard simplex methods perform the same sequence of column, and row choices and take the same number of iterations. This allows us to compare performance of the two approaches by comparing the average time per iteration rather than the total running time. This is very convenient because performance models of the time per iteration are much easier to come by than for total time. In other cases, for example, in comparing performance for different column choice rules, total time must be compared since the number of iterations may be quite different.

As will be discussed below, there are alternative ways to choose a column but the classical rule is most commonly employed and has the lowest computational cost.

The simplex method consists of three basic steps:
High-level serial algorithm

a. Column choice
b. Row choice
c. Pivot

Updating any of the representations is, at most, of order m^2 average work. On the other hand, pivoting in the standard method on the explicit representation of the dictionary takes order *mn* work. Thus for high aspect ratios, the standard method takes more work.

The following are the costs of the steps:

Cost of classical column choice for tableau method: nothing - $\hat{c}_j$ is kept explicitly for all j. (The ratio test cost is being ignored.)

Cost of classical column choice for revised method: $m(n-m)10\%$ There are m rows and n-m nonbasic columns. We assume partial pricing of an average 10% of the columns. Of course the actual percentage depends on the settings of the revised method.

Cost of row choice for tableau method: nothing – column $\widehat{A_j}$ is kept explicitly for all j. (The comparison for minimum is being ignored.)

Cost of row choice for revised method: m^2 to generate the entering column $\widehat{A_j}$.

Cost of pivot for tableau method: *m(n-m)*

Cost of pivot for revised method: update of the B^{-1} matrix: m^2 - this holds on average with LU decomposition too

To summarize:

Cost	Classical column choice rule	Row choice	Pivot
Tableau	-	-	$m(n\text{-}m)$
Revised	$m(n-m)10\%$	m^2	m^2

Table 2: Cost Comparison of Revised and Tableau for classical column choice rule and pivot

3. Use of parallel computing is advantageous to the tableau method

As reported in Yarmish and Van Slyke [32], the tableau method keeps all columns explicitly whether the column correspond to the variables currently in in the basis or not. It is easy to distribute the column amongst many machines.

We described a relatively straightforward parallelization scheme within the standard simplex method involves dividing up the columns amongst many processors. Instead of the three basic steps of the last section we would have five basic steps:

High-level parallel algorithm

a. Column choice – each processor will "price out" its columns and choose a locally best column (Computation).
b. Communication amongst the processors of the local best columns. All that is sent is the pricing value (a number) of the processor's best column. At the end of this step each processor will know which processor is the "winner" and has the global column choice (Communication).
c. Row choice by the winning column (Computation).
d. A broadcast of the winning processor's winning column and choice of row (Communication).
e. A simultaneous pivot by all processors on their columns (Computation).

For more details and analysis of these steps see Yarmish and Van Slyke [32].

4. Alternative Column Choice Rules for the standard simplex method

4.1 Three possible column choice rules

Any eligible non-basic variable may be chosen in the column choice step of the simplex method. We discuss three approaches to picking the particular non-basic variable.

4.1.1 The classical column choice rule

The original rule used by Dantzig was to choose the eligible c'_j in the current dictionary with the largest absolute value. This selects the non-basic variable that gives the largest improvement in the objective function per unit change of the non-basic variable. This criterion is very simple and straightforward to compute. In contrast to some of other methods, the column is chosen without looking at any of the coefficients, a'_{ij}. However, it is has the undesirable feature that by rescaling the variables you can cause any eligible column to be chosen.

4.1.2 The greatest change rule

For each eligible column, perform the **row** choice step of the simplex method and then compute the improvement in the objective that would result if the column were chosen, and use this as the column choice rule. This is called the *greatest change* criterion. It takes even more work than the steepest edge rule. The payoff seems no better than for the steepest edge rule, so it is rarely used (see Section 9.2). Nevertheless, this method can be implemented easily in standard implementations.

4.1.3 The steepest edge column choice rule

The dependence of the column choice on scaling of the classical method can be removed by normalizing the value of c'_j by the length of the column in the current dictionary corresponding to the non-basic variable j. Applying the steepest edge rule requires more work per iteration for both standard and revised methods. In both cases, for each eligible column one has to compute the norm of the column **in terms of the current basis**. In addition, in revised methods, one does not have readily at hand the current representation a'_{ij}. This would seem to rule out the steepest edge rule for revised implementations; however, clever recursive computations can be used to implement the rule with modest cost [11]. The *Devex* rule of Harris [18] is another scheme for the revised method that approximates the steepest edge criterion efficiently. In any case, the standard method has the needed coefficients readily available.

5. Computational difficulty of using steepest edge column choice rules especially for the revised method

5.1 The revised simplex does not explicitly keep all information necessary for these rules

As the revised simplex method became the de facto method the cost per iteration of these rules became even less attractive. This is due to the fact that the nonbasic columns, which are not explicit in the revised method, need recalculation for each column.

5.2 Partial pricing

Another downside of using the steepest edge method is the common technique of partial pricing used in the revised method. The idea behind partial pricing is to avoid the costly pricing out of so many columns. In order to take advantage of the steepest edge method full pricing needs to be employed. As explained in more detail below employing partial pricing would negate the iteration-lowering effects of the steepest edge method. This directly impacts the extra cost per iteration of using the steepest edge rule vs. the classical rule as explained below. In the context of the revised method the cost per iteration necessary for these alternate column choice rules, including the steepest edge rule, was therefore prohibitive and negated the reduction of the iteration count.

5.3 Update methods for the revised steepest edge are helpful but still costly

More recently efficient update formulas for the steepest edge rule, within the revised simplex method, have been developed. The first one, Devex, approximates the norms necessary for the steepest edge method. Subsequently exact update formulas for both the primal revised method [12] and for the dual revised method [11] have been introduced. Although the computational cost has been mitigated somewhat by Forrest and Goldfarb's updating method for the steepest edge rule, there is still significant extra computation necessary in order to use the steepest edge rule.

6. Analysis of additional cost per iteration for use of the steepest edge rule revised vs. full tableau method

6.1 Steepest edge update formula for the revised method

In this section we go through an analysis of the steepest edge update method of Forrest and Goldfarb [11] and assume some familiarity with the revised method in matrix notation. The key element to keep in mind is that the full tableau is not kept explicitly. All columns that correspond to variables not currently in the basis must be recomputed from B^{-1} and from the initial column. The following notation roughly follows Nash and Sofer [23].

B is defined as the square matrix consisting of the initial column of the current basic variables.
A_j is the initial column of the j^{th} nonbasic variable
$\widehat{A_j}$ is the current column of the j^{th} variable. This column is not kept explicitly in the revised method.
Every variable with a 'hat' is value of the current matrix which is not kept explicitly in the revised method but is kept explicitly in the tableau method.

Steepest edge column choice rule:

$$\min_j \frac{\widehat{c_j}}{||\widehat{A_J}||}$$

The norm

$$\left|\left|\widehat{A_J}\right|\right| = \sqrt{\widehat{a_{1J}}^2 + \widehat{a_{2J}}^2 + \cdots \widehat{a_{mJ}}^2}$$

Define the scalar

$$\gamma_J = \widehat{a_{1J}}^2 + \widehat{a_{2J}}^2 + \cdots \widehat{a_{mJ}}^2$$

The steepest edge update formula for a new γ_j is:

$$\gamma_j' = \gamma_j + \frac{(B^{-1})^T e_s A_j}{\widehat{a_{st}}}(1+\gamma_t) - 2\frac{(B^{-1})^T e_s A_j}{\widehat{a_{st}}} A_j (B^{-1})^T \widehat{A}_j$$

Where γ_j' and γ_j refer to the updated and previous pivot respectively (the 'hat' is left off). S within refers to the leaving variable and t refers to the entering variable.

6.2 Cost analysis per iteration classical vs. steepest edge column choice rule for tableau and revised methods

Cost of classical column choice for tableau method: nothing - $\widehat{c_j}$ is kept explicitly for all j. (The comparison for minimum is being ignored.)

Cost of classical column choice for revised method: $(m * n)10\%$ assuming partial pricing of an average 10% of the columns.

Cost of steepest edge method for tableau method: $(m + \text{sqrt}) * n = mn + n * sqrt$

Cost of steepest edge method for revised method:

- m^2+m to calculate $(B^{-1})^T e_s A_j$,
- m to calculate $\widehat{a_{st}}$,
- m^2 to do the division and
- m^2+m to calculate $A_j (B^{-1})^T \widehat{A}_j$

for a total of

$(2(m^2+m)+m^2+m+sqrt)n = 3m^2n + 2mn + n * sqrt$

To summarize:

Cost	Classical rule	Steepest edge rule
Tableau	-	$mn + n * sqrt$
Revised	$(mn)10\%$	$3m^2n + 2mn + n * \text{sqrt}$

Table 3: Cost Comparison of Revised and Tableau for classical and steepest edge column choice rule

It is important to note that partial pricing is used to great effect in the revised method when using the classical rule but cannot be used with the steepest edge without severely mitigation the advantage of the steepest edge method. This is because, as briefly mentioned earlier, the whole idea of the steepest edge method is to pick a good choice for an entering variable in order to reduce the iteration count. If partial pricing is used this advantage would be negatively affected. The reason partial pricing is reasonable when using the classical column choice rule, on the other hand, is precisely that it is not as good a heuristic choice and therefore it is not as negatively affected by partial pricing.

In summary, when one switches to the steepest edge method the extra cost to the revised over the tableau method per iteration is:

$(3m^2\text{n} + 2\text{mn} + \text{n} * \text{sqrt}) - (m\text{n} + \text{n} * sqrt) - (\text{mn})10\%$

$= \ 3m^2\text{n} + (\text{mn})90\%$

It must be pointed out that the revised method can take advantage of sparsity for this extra calculation.

7. Experimental comparison of column choice rules for the revised method

Eigen [10] did a comparison of column choice rules including the classical rule, the exact steepest edge rule and two forms of Devex, which are steepest edge approximations. The comparisons were done for the Netlib problems. He utilized the lpsolve code of Berkelaar [3] plus his own implementation of a second devex method.

He found that the classical rule does fairly well and fared only slightly worse than devex. Analysis of his data shows that there is a significant gain in terms of iterations which is not reflected in the time savings but is much more modest.

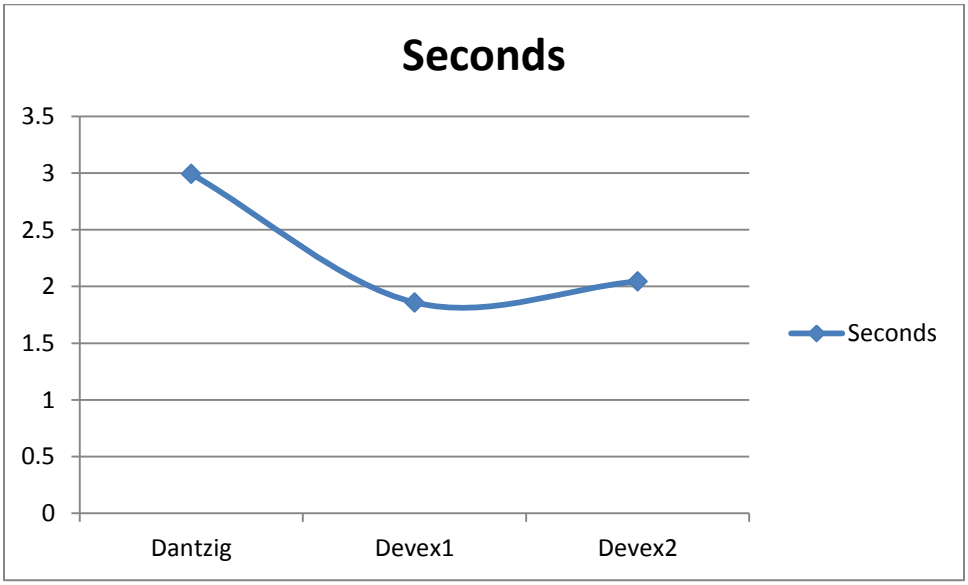

Figure 1

Figure 2

We used the iteration and timing data of the netlib problems as reported in Eigen [10] to generate figure 1 and figure 2. We did not include in the data for these figures problems that did not have data for the Dantzig (classical), devex1, devex2 or the steepest edge column choice rules. Figure 1 and figure 2 show the average run time and the average number of iterations respectively. As can be seen by inspection although the number of iterations used by devex went down to only 16% of the number of iterations vis a vis the classical rule, the actual timing was only reduced to 63% of the classical rule. The exact steepest edge method, although not included in these figures, does even worse than devex in terms of both iteration count and computation time.

It is worth keeping in mind that the Netlib problems are not average problems but were uploaded to the repository specifically due to their difficult nature and one has to be careful when extrapolating to other problems.

At least for this data set, the Devex approximation code did better than the exact steepest edge update method.

What is important, for us, is to note that, for the revised method, the extra computational cost per iteration significantly reduces computational gains from the reduction of the iteration count.

8. Experimental Configuration

We performed experiments on problems from the Netlib library, and synthetic problems. Both pose difficulties. The Netlib problems are not at all typical. Many of them have been submitted because of "nasty" features that make them thorough tests of linear programming codes. See, for example, Ben-Tal and Nemirovski [2] for a discussion of this. Moreover, the problems are very sparse. Finally, we wished to determine the performance of retroLP as a function of problem parameters, particularly density. To be able to control for the problem parameters, synthetic problems are convenient, but they may have covert features that make them much easier (or much

harder) than "typical" problems. We used multiple generators to try to minimize this potential problem.

It is important to note that for the sake of comparison partial pricing, scaling, and basis "crashing," were disabled for these tests.

8.1 Test Sets

Netlib contains problems for testing linear programming codes [www.netlib.org/lp/data, 1996]. While our program successfully ran all the Netlib problems, we used as our test set the 30 densest problems. These include all problems with density above 2.5%
We used three synthetic program generators. Each takes as input, m = number of rows, n = number of columns, d = the density of the non-zero coefficients ($0 < d \leq 1$), and *seed* = the seed for the random number generator. All the constraints are of the less than or equal type. Whether a coefficient, a_{ij}, of the constraint matrix is non-zero (or zero) is determined randomly with probability d. For one of the generators, the value of a non-zero coefficient is chosen at random, uniformly between -1 and 1. The objective coefficients are generated randomly between -1 and 1 (the sparsity condition does not apply to the objective). The variables are constrained to be between $-m$ and m. The constraints are constrained to range between -1 and 1. Since, setting all variables to 0 is feasible, no Phase 1 is required. The other two generators are similar.

8.2 MINOS

We use MINOS 5.5 [22] as a representative implementation of the revised simplex method. We installed it to run in the same environment as retroLP. This allowed us to make reasonable comparisons between the standard and revised simplex methods. The purpose of these comparisons is not so much to compare running times but to examine the relative behavior of these approaches as the parameters of interest, primarily density, are varied. In general we used the default parameters with MINOS with a few, significant exceptions designed to make MINOS more comparable with retroLP. For the sake of comparison partial pricing, scaling, and basis "crashing," were disabled for these tests.

9. Experimental Results: Density crossover points of tableau vs. revised methods for classical and steepest edge column choice rules

9.1 Eligible Columns

When using steepest edge, or greatest change column choice rules, the amount of work in "pricing out" a column differs dramatically depending on whether the column is eligible or not. To determine eligibility, basically two comparisons are needed; however, if the column is, in fact, eligible an additional order m computations are needed. So for accurate performance analysis it is useful to be able to estimate the fraction of columns that are eligible. When using greatest change column choice rule on the 30 Netlib problems, the fraction of columns that are eligible ranges from 4.6% to 42.9%. A simple average of the percentages is 23.18% while a weighted average resulted in 42.9% (one problem ran for very many iterations). For steepest edge the range was 4.6% to 56.44%. The simple average was 26.15% and the weighted average 40.74%. So it is rare that one needs to even consider half the columns in detail.

9.2 Iterations by Column Choice Rules

An important factor in performance is the column choice rule used. Generally, there is a tradeoff between the number of iterations using a rule and the computational effort it takes to apply the rule to the column. The number of iterations resulting from the use of a particular rule depends only on the problem, while the computational effort to apply the rule depends on the specific implementation as well. Most dramatically the effort depends on whether a standard or revised method is used, but choices of programming languages, skill of coders, and the particular hardware used is important also.

Based on experimentation we found that the ratio of the number of iterations using the greatest change rule to the number using the classical rule ranges from 0.378 to 5.465. The simple average of the 30 ratios is 1.140, and the average weighted by the number of iterations is 1.053.

For the steepest edge, the range was 0.318 to 1.209. The simple and weighted averages were 0.800 and 0.620, respectively. The averages were computed considering only major iterations, but the results were essentially the same based on all iterations. *Note that the ratio .8 for the steepest edge method is similar to the 20% lower iteration count for the steepest edge over the classical column choice rule reported above from Eigen's data.*

Compared to steepest edge, rarely does the classical method result in fewer iterations, and then only slightly, see also Forrest & Goldfarb [11]. The greatest change rule, on the other hand, seems to offer little benefit compared to the classical method so we did not consider it further.

9.3 Performance Models

For the standard simplex method with the classical column choice rule, the time spent in pivoting can be over 95%. Fortunately the pivot routine is rather simple. This makes performance analysis straightforward. Virtually, all the instructions in *pivot* are of the form:

$A_{ij}=A_{ij}*t$ where A contains the constraint tableau and t is the unit time for a multiplication

With the steepest edge column choice rule, the column choice time becomes significant. Typically about 75% of the time might be spent on pivoting and 25% on column choice. Usually the other parts of the program use little time. Because the dynamics of the column choice procedure is more complex than pivoting, the timing approach used in analyzing the classical column choice rule is difficult to apply.

In order to predict the time for the steepest edge method we experimentally calculated Unit Times for pivots and steepest edge column choice directly from runs on the test problems UT_{se} and UT_{p}.

For unit pivot time, we run an *n x m* size test problem and perform p pivots without the column choice or row choice steps.

For unit steepest edge time we ran the steepest edge code p times on a column of size m, where p is an arbitrary number. We then divided the total time by m*p to obtain a unit steepest edge time UT_{se}.
We then end up with a performance model for retroLP of the following form:

$$T = p[(m+2)n+1]UT_p + c_e(m+1)UT_{se}$$

where p is the number of pivots, c_e is the number of eligible columns evaluated using the steepest edge rule, UT_p is the unit time for major pivots, and UT_{se} is the unit time for the steepest edge evaluation for eligible columns. Most accurately, T accounts for the column choice plus the pivot time; however, the other contributions are generally quite small and T offers a good approximation to total time. When using the classical column choice rule, the last term $c_e(m+1)UT_{se}$ is not used. *Figure* shows how well the actual time spent in pivoting and column choice for the Netlib problems (excluding the three largest) using steepest edge column choice compared with the predicted time.

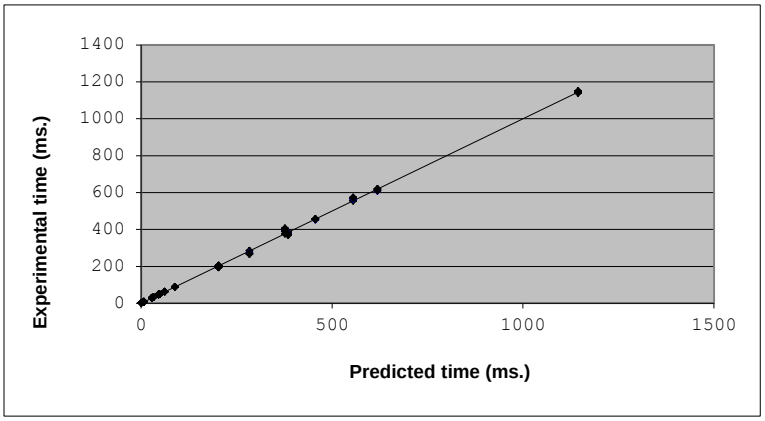

Figure 3: Pivot and Column Choice Time -- Predicted and Actual

9.4 Comparison of Revised and Tableau Simplex Methods

We first compare retroLP and MINOS when both use the classical column choice rule. Next we compare retroLP using steepest edge with MINOS using the classical rule (MINOS does not support steepest edge). In this latter case, for the first time, we must base our comparisons on total running time. These tests were on synthetic linear programs with m=500, and n=1,000. For each data point three different problem generators with three different seeds for a total of nine combinations were run.

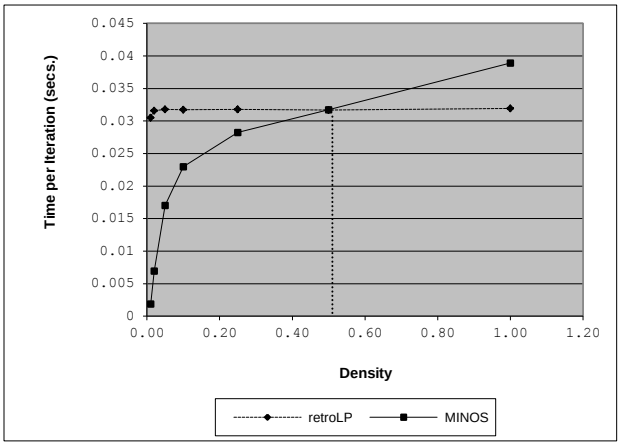

Figure 4: Comparison of tableau and revised Iteration Time vs. Density

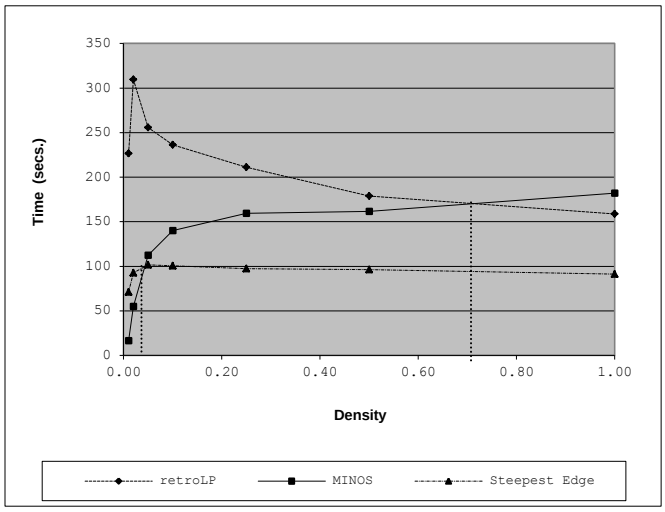

Figure 5: Comparison of Total Running Time

In *Figure 4* we see that the time per iteration of the tableau method is essentially independent of density, while the iteration time of the revised goes up with density. The crossover point is about 50% density.

Figure is a comparison of total running time for the tableau method using both classical column choice, and steepest edge, and the revised method using classical column choice. The breakeven for retroLP and MINOS both using classical column choice is at about 72% density. The breakeven for MINOS using classical column choice and retroLP using steepest edge has gone down to about 5% density.

10. Summary and Conclusions

In this paper we demonstrated the advantage of using the steepest edge based rules for the tableau method. It is well known that this rule, on average, significantly reduces the number of iterations of the simplex algorithm. The advantage for the revised method, though, is mostly offset by the extra computation required to implement it. On the other hand the tableau method, because it naturally keeps the full columns of the nonbasic variables does not require this computation.

We began with a basic review of the simplex method and then discussed alternate column choice rules. From there did a cost analysis for both the tableau and revised method.

We showed that the density crossover point when we use steepest edge column choice rules was reduced to 5% down from about 72% when the classical rule was used for the tableau method, see Table 1 above. Although we did not implement steepest edge on the revised method, we discussed results that demonstrate that the advantages to the revised method are largely offset by the extra computation necessary.

What was not shown and would be interesting future work is to combine a distributed tableau with the steepest edge method. We has previously show the density crossover point is significantly lowered using distributed processing – down to below 10% from around 72% in our examples. The steepest edge column choice rules should lend itself to the same speedup. This promises to further reduce the density crossover point.

References

[1] Bartels, R.H. and Golub, G.H. 1969. The simplex method of linear programming using LU decomposition. *Communications of the ACM*. 12, 5 (1969), 266–268.

[2] BenTal, A. and Nemirovski, A. 2000. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*. 88, 3 (2000), 411–424.

[3] Berkelaar, M. et al. 2006. lpsolve 5.5, Open source (Mixed-Integer) Linear Programming system Software. *Eindhoven University of Technology, http://sourceforge. net/projects/lpsolve*. (2006).

[4] Bixby, R.E. and Martin, A. 2000. Parallelizing the Dual Simplex Method. *INFORMS Journal on Computing*. 12, 1 (Winter 2000), 45–56.

[5] Bürger, M. et al. 2012. A distributed simplex algorithm for degenerate linear programs and multi-agent assignments. *Automatica*. (2012).

[6] Chen, S.S. et al. 2001. Atomic decomposition by basis pursuit. *SIAM review*. 43, 1 (2001), 129–159.

[7] Chvatal, V. 1983. *Linear programming*. Macmillan.

[8] Dantzig, G.B. and Orchard-Hays, W. 1954. The product form for the inverse in the simplex method. *Mathematical Tables and Other Aids to Computation*. (1954), 64–67.

[9] Eckstein, J. et al. 1995. Data-parallel implementations of dense simplex methods on the connection machine CM-2. *ORSA Journal on Computing*. 7, 4 (1995), 402–416.

[10] Eigen, D. 2011. Pivot Rules for the Simplex Method.

[11] Forrest, J.J. and Goldfarb, D. 1992. Steepest-edge simplex algorithms for linear programming. *Mathematical programming*. 57, 1-3 (1992), 341–374.

[12] Goldfarb, D. and Reid, J.K. 1977. A practicable steepest-edge simplex algorithm. *Mathematical Programming*. 12, 1 (1977), 361–371.

[13] H.W. Kuhn and Quandt, R.E. 1963. An experimental study of the simplex method. *Proceedings of symposia in applied mathematics*. Am. Math. Soc., Providence, RI.

[14] Haksever, C. 1993. New evidence on the efficiency of the steepest-edge simplex algorithm. *Computers & Industrial Engineering*. 24, 3 (Jul. 1993), 401–412.

[15] Hall, J. and Huangfu, Q. 2012. Promoting hyper-sparsity in the revised simplex method. (2012).

[16] Hall, J. and Smith, E. 2010. A parallel revised simplex solver for large scale block angular LP problems. (2010).

[17] Hall, J.A.J. 2010. Towards a practical parallelisation of the simplex method. *Computational Management Science*. 7, 2 (2010), 139–170.
[18] Harris, P.M. 1973. Pivot selection methods of the Devex LP code. *Mathematical programming*. 5, 1 (1973), 1–28.
[19] Lalami, M.E. et al. 2011. Efficient implementation of the simplex method on a CPU-GPU system. *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on* (2011), 1999–2006.
[20] Lalami, M.E. et al. 2011. Multi gpu implementation of the simplex algorithm. *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on* (2011), 179–186.
[21] Liu, C.-M. 2002. A Primal-dual Steepest-edge Method for Even-flow Harvest Scheduling Problems. *International Transactions in Operational Research*. 9, 1 (2002), 33–50.
[22] Murtagh, B.A. and Saunders, M.A. 1998. *MINOS 5.5 user's guide. Report SOL 83-20R, Dept of Operations Research*. Stanford University.
[23] Nash, S.G. and Sofer, A. 1996. *Linear and nonlinear programming*. McGraw-Hill New York.
[24] Pan, P. 2010. A FAST SIMPLEX ALGORITHM FOR LINEAR PROGRAMMING. *Journal of Computational Mathematics*. 28, 6 (2010), 837–847.
[25] Ploskas, N. et al. 2009. A parallel implementation of an exterior point algorithm for linear programming problems. *Proc. of the 9th Balkan Conference on Operational Research* (2009).
[26] Reid, J.K. 1982. A sparsity-exploiting variant of the Bartels—Golub decomposition for linear programming bases. *Mathematical Programming*. 24, 1 (1982), 55–69.
[27] Sloan, S.W. 1988. A steepest edge active set algorithm for solving sparse linear programming problems. *International Journal for Numerical Methods in Engineering*. 26, 12 (1988), 2671–2685.
[28] Terlaky, T. and Zhang, S. 1993. Pivot rules for linear programming: a survey on recent theoretical developments. *Annals of Operations Research*. 46, 1 (1993), 203–233.
[29] Thomadakis, M.E. and Liu, J.-C. 1996. An E cient Steepest-Edge Simplex Algorithm for SIMD Computers. (1996).
[30] Wolfe, P. and Cutler, L. 1963. Experiments in linear programming. *Recent advances in mathematical programming*. McGraw-Hill, New York.
[31] Yarmish, G. 2007. Wavelet decomposition via the standard tableau simplex method of linear programming. *WSEAS TRANSACTIONS ON MATHEMATICS*. 6, 1 (2007), 170.
[32] Yarmish, G. and Slyke, R.V. 2009. A distributed, scaleable simplex method. *The Journal of Supercomputing*. 49, 3 (Sep. 2009), 373–381.