A General Pricing Scheme for the Simplex Method

István Maros

Department of Computing

Imperial College, London

i.maros@ic.ac.uk

Departmental Technical Report 2001/3

ISSN 1469–4174

March 2001

Contents

Abstract

Pricing is a term in the simplex method for linear programming used to refer to the step of checking the reduced costs of nonbasic variables. If they are all of the ‘right sign’ the current basis (and solution) is optimal, if not, this procedure selects a candidate vector that looks profitable for inclusion in the basis. While theoretically the choice of any profitable vector will lead to a finite termination (provided degeneracy is handled properly) but the number of iterations until termination depends very heavily on the actual choice (which is defined by the selection rule applied). Pricing has long been an area of heuristics to help make better selection. As a result, many different and sophisticated pricing strategies have been developed, implemented and tested. So far none of them is known to be dominating all others in all cases. Therefore, advanced simplex solvers need to be equipped with many strategies so that the most suitable one can be activated for each individual problem instance. In this paper we present a general pricing scheme. It creates a large flexibility in pricing. It is controlled by three parameters. With different settings of the parameters many of the known strategies can be reproduced as special cases. At the same time, the framework makes it possible to define new strategies or variants of them. The scheme is equally applicable to general and network simplex algorithms.

1 Introduction

Pricing is certainly an evergreen research topic in the simplex method for linear programming (LP). This step does two things. It checks whether the optimality conditions for the nonbasic variables are satisfied, and if not it chooses a violating variable to enter the basis. The selected variable has the potential of improving the current solution. According to the theory the simplex method terminates in a finite number of iterations regardless of which profitable candidate is chosen (if degeneracy is treated properly). On the other hand, the total computational effort to solve a problem heavily depends on this choice.

The importance of a good selection rule has long been recognized. While many pricing strategies have been worked out none of them proved to be the best in the sense that it dominates all others on all problems. Important ideas have been put forward by Dantzig [4], Orchard-Hays [12], Harris [9], Benichou at al. [1], Bland [2], Goldfarb and Reid [7], Greenberg [8], Cunningham [3], and Forrest and Goldfarb [5]. Fourer [6] discusses pricing designed for staircase linear programs. Advanced simplex solvers are equipped with several pricing strategies and the user or the solution algorithm itself can decide which one to activate for any given problem instance.

Maros [10] described a pricing procedure for structured network problems in the network simplex method. In this paper we elaborate on this work and propose a general pricing framework. It creates a large flexibility and seems to be practical for structured LP problems but also can be beneficial for others, too. It incorporates many of the know pricing strategies as special cases.

The rest of the paper is organized as follows. Section 2 sketches the revised simplex method for further reference. Some characteristics of structured problems are discussed in section 3. Section 4 gives an annotated list of the best known pricing strategies. The proposed pricing scheme is introduced and discussed in section 5 which is followed by some concluding remarks in section 6.

2 Problem statement and algorithmic framework

We consider the primal LP problem in the following general form:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x}, \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, && (1) \\ & \mathbf{l} \le \mathbf{x} \le \mathbf{u}, && (2) \end{aligned}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{c}$, $\mathbf{x}$, $\mathbf{l}$, $\mathbf{u} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$. Arbitrary components of $\mathbf{l}$ and $\mathbf{u}$ can be $-\infty$ or $+\infty$, respectively. Without loss of generality, we can assume that all finite lower bounds in (2) are shifted to zero.

In formulating LP problems, it is typical that $\mathbf{A}$ contains a unit matrix $\mathbf{I}$, $\mathbf{A} = [\mathbf{I}, \bar{\mathbf{A}}]$, as each row is assigned a *logical variable* to make the constraints equality and, therefore, $\mathbf{A}$ is of full row rank.

Let B denote a basis to (1) (which is actually the set of indices of basic variables) and $\mathbf{B}$ the corresponding basis matrix of $\mathbf{A}$. A similar notation for the nonbasic part of $\mathbf{A}$ is N and $\mathbf{N}$. The matrix and the vectors are partitioned as $\mathbf{A} = [\mathbf{B} \mid \mathbf{N}]$, $\mathbf{x} = [\mathbf{x}_B, \mathbf{x}_N]$ and $\mathbf{c} = [\mathbf{c}_B, \mathbf{c}_N]$. The constraints in (1) can be rewritten as $\mathbf{B}\mathbf{x}_B + \mathbf{N}\mathbf{x}_N = \mathbf{b}$, or equivalently, $\mathbf{x}_B = \mathbf{B}^{-1}(\mathbf{b} - \mathbf{N}\mathbf{x}_N)$. Variables in N are either at zero or upper bound or some ‘superbasic’ value.

As large scale LP problems exhibit increasingly high degree of sparsity the only realistic version of the simplex algorithm is the sparsity exploiting implementation of the revised simplex method, often referred to as SSX.

The iteration cycle of the computational version of the primal SSX consists of the following steps.

Step 1. Compute **simplex multiplier** $\boldsymbol{\pi}^T = \mathbf{v}^T\mathbf{B}^{-1}$ using a form vector $\mathbf{v}$ (different for Phase-1 and Phase-2).

Step 2. **Check optimality** (also known as *Choose Column* or *PRICE*): compute reduced costs $d_j = c_j - \boldsymbol{\pi}^T\mathbf{a}_j$ if in Phase-2 or $d_j = \boldsymbol{\pi}^T\mathbf{a}_j$ if in Phase-1, for $j \in N$. If all of them satisfy their optimality condition, exit with "optimal solution" or "problem infeasible". Otherwise, there is at least one variable that can enter the basis. We choose one from among the candidates using some (simple or complex) criterion. Its subscript is denoted by q.

Step 3. **Transform** the column vector $\mathbf{a}_q$ of the improving candidate: $\boldsymbol{\alpha}_q = \mathbf{B}^{-1}\mathbf{a}_q$.

Step 4. **Choose row** (also known as *pivot step* or *ratio test*): determine the outgoing variable or bound swap or conclude that "solution is unbounded" and exit.

Step 5. **Update** solution, basis inverse and any other changing information; do reinversion (refactorization) if necessary.

There are some variants of SSX where a row vector of reduced costs is maintained. In this case Step 2 is simpler but Step 5 becomes computationally more intensive.

3 Structure in LP problems

Real life LP problems are often very large in size and usually have some structure. This is because they typically represent multi-period, multi-location, dynamic or stochastic models.

Structure can be exploited in SSX to speed up iterations or to make them more effective. Different levels of sophistication can be applied in the design of such strategies. If there is more information available about the structure then better pricing can be achieved (c.f. [6]). The industry standard LP input format (MPS format) is not defined to carry such information. Therefore, general purpose simplex solvers are usually not prepared to take advantage of structure directly. We come back to this issue in section 6.

For now, we assume that the variables of the LP problem exhibit some identifiable structure. It means that variables can be grouped into clusters according to the nonzero pattern of their column vectors. Vectors in a cluster are more related to each other than vectors in different clusters. It is quite likely that each cluster will have representatives in every basis and, therefore, in the optimal solution.

4 Pricing in the simplex method

Column selection is often referred to as the PRICE step (see section 2). During pricing, the d_j reduced costs of the nonbasic variables are computed. If there is at least one that violates its optimality condition it is a candidate to enter the basis. Since usually there are many improving candidates the goal is to select the best one in some sense. What can this sense be? An obvious interpretation is to call 'best' at any given iteration the one which results in a least number of iterations. As SSX provides only local information it is impossible to identify the best in this sense. Another possibility is to call 'best' the one which leads to the least computational effort (shortest solution time). This is again an unachievable goal. The most that can be done is to try to approximate one of these goals. It is usually the minimization of the computational effort that is aimed at by different pricing heuristics.

PRICE is computationally an expensive step because of the many dot products that have to be calculated for some or all the nonbasic columns of the $\mathbf{A}$ matrix. This equally applies to the case when the nonbasic d_j's are kept explicitly and updated at each basis change.

It is well known that the actual magnitude of d_j's is scale dependent and a profitable variable with the largest d_j usually does not lead to the largest improvement in the objective function. While the scaling procedure applied to LP problems prior to solving tries to consolidate the magnitudes it does not resolve this problem. There are some pricing techniques that attempt to overcome this difficulty by dynamically determining some weight factors for nonbasic variables. The d_j's are normalized by these weights before being compared. Below we enumerate some of the best known PRICE strategies for SSX.

1. *First improving candidate.* The first $\mathbf{a}_j,\ j \in N$ with nonoptimal d_j is selected. This is the cheapest criterion (part of Bland's rule [2]) but it usually leads to a very large number of iterations. Therefore, it is mostly of theoretical importance and is not used in practice.

2. *Dantzig rule* [4]. In this option all nonbasic variables are checked (*full pricing*) and one which violates its optimality condition the most is selected. This rule is quite expensive (dot products with all nonbasic variables) but, overall, is considerably better than the previous method.

3. *Partial pricing.* To alleviate the computational burden, only a part of the nonbasic variables is scanned and the best candidate from this part is selected [12]. In the next step, the next part is scanned, and so on. If the partition is chosen initially and kept fixed then this is known as *static partial pricing.* In contrast, the parts of matrix $\mathbf{A}$ may dynamically be redefined during the SSX iterations; this is known as *dynamic partial pricing* [1].

4. *Multiple pricing.* Some of the most profitable candidates (in terms of the magnitude of d_j) are selected during one scanning pass (*major iteration*, [12]). They are updated and a *suboptimization* is performed involving the current basis and the selected candidates using the criterion of greatest improvement. During these *minor iterations* the update of the selected candidates is inexpensive but some updated columns may become nonprofitable even before using them. Still, this technique is successful and is included in many solvers.

5. *Sectional pricing* [8]. It can be viewed as a kind of partial pricing. LP models often exhibit some structure which is characterized by the presence of sections or clusters of variables. In this case pricing can operate more effectively by choosing candidates from different sections during a multiple pricing step. This increases the possibility that the selected candidates are not or just loosely related and most of them will remain candidates during the minor iterations.

6. *Steepest edge.* The magnitude of d_j shows the rate of change (scale dependent) but not how far the selected variable can go. Hence, it is not a good indicator of the achievable progress in the objective function. Normalized pricings attempt to estimate the relative merit of the improving candidates by evaluating d_j in a fixed

framework. This is achieved by scaling the computed d_j by a scale factor before comparison. Scale factors are updated after every basis change.

Steepest edge [5] is the most powerful normalized pricing algorithm but it is computationally the most expensive. It is a full pricing and does not adapt to the multiple pricing scheme. It can dramatically reduce the total number of iterations but the work per iteration can be so large that there can be no computational gain. However, in certain problem instances this strategy is clearly superior to any other method. It is used in the dual simplex more frequently because it requires less extra computations there.

7. *Devex.* Introduced by Paula Harris [9], this was the first effective normalized pricing; today we see it as an approximation to steepest edge. It requires less work per iteration but still can reduce the number of iterations quite considerably. In a number of cases it results in the reduction of the overall computational effort and is considered a useful tool for the primal SSX but it is also easily adaptable to the dual. It is a full pricing and is not suitable for multiple pricing.

8. *Dynamic scaling* [1]. This is an inexpensive approximation to the steepest edge pricing. Its advantage is its suitability for partial and multiple pricing. Its effectiveness is, however, unproven for many problem instances even though there is evidence of its usefulness in some cases [11].

5 A general pricing scheme

In this section we introduce a pricing scheme that creates a great flexibility and contains many known pricing strategies as special cases. It also makes it possible to define new pricing strategies.

Let the index set of all variables in the problem be denoted by S and let K denote the number of disjoint clusters of variables. The main characteristics of variables in a cluster is that they are somehow related to each other. Let S_k, $k = 1, \dots, K$, denote the index set of variables in cluster k with $n_k = |S_k| > 0$ denoting its cardinality. The set of indices of all variables is thus

$$S = \bigcup_{k=1}^{K} S_k, \qquad S_i \cap S_j = \emptyset, \quad \text{if} \quad i \neq j.$$

Obviously, $|S| = n$.

We consider a fixed circular ordering of the variables within each cluster, and also a fixed circular ordering of the clusters. It can be assumed that members of a cluster are located contiguously. If not, a linked list structure can be created and the variables can be accessed by some fixed order in this list.

Common sense suggests and experience shows that if a properly selected vector from a cluster enters the basis at a given iteration then it is unlikely that good candidates in the same cluster (or in clusters that have also recently been visited) can be found during the next pricing pass. This can be interpreted in such a way that if the—locally—best vector enters the basis then, as a consequence of the correlation among the members of the cluster, the other candidates usually lose their attractiveness as they have become represented in the current solution by their 'best' candidate. If so, we can avoid superfluous pricing of non-improving or little-improving vectors by starting to look for new candidates in a different cluster(s). At the same time, we do not have to visit all the clusters if a sufficient number of profitable candidates have been found (partial pricing of the clusters). Furthermore, it is not necessary to make a complete scan of a cluster either. We can stop scanning it if an acceptable number of candidates have been found there (partial pricing of a cluster).

On the basis of what we have said so far, we can introduce the following pricing (column selection) algorithm, referenced as **SIMPRI** (for Simplex Pricing) in the sequel.

The main parameters that control SIMPRI are:

- K : the number of clusters (defined above),
- P : the number of clusters to be scanned in one iteration ($1 \leq P \leq K$),

- R : the number of improving vectors to be found in one cluster (this number can be R_k, if we allow it to be different for each cluster).

The i-th element of cluster k in the fixed circular traversing order will be denoted by e_i^k. Two pointers are also needed: k will point to a cluster, while i_k will point to an element in cluster k.

In a general step of the algorithm pricing stopped in S_k at $e_{i_k}^k$. The next pricing will start in the next cluster S_{k+1} at the next element $e_{i_{k+1}+1}^{k+1}$ to come after the one where pricing stopped at the last visit in cluster S_{k+1}. It means that if we increment k by 1 (mod K) and i_k by 1 (mod n_k) we can start pricing formally in S_k at $e_{i_k}^k$.

To describe the complete algorithm in which this general step is imbedded, some auxiliary variables have to be introduced:

- r is the number of improving variables found so far within a cluster during the current pass of SIMPRI,
- s counts the number of clusters scanned,
- q counts the number of vectors interrogated in a cluster
- $d_{\max}$ is the largest reduced cost found so far,
- j is the absolute index ($1 \le j \le n$) of $d_{\max}$.

Algorithm **SIMPRI**:

Step 0. *Setup.* Initialize

pointers: $k = K,\ i_k = n_k,\ k = 1, \ldots, K$,

counts and markers: $s = 0$, $d_{\max} = 0$, $j = 0$.

Step 1. *Cluster count.* Increment s. If $s > K$ go to Step 7.

Step 2. *Cluster initialization.* Increment k (mod K). Set $r = 0$, $q = 0$.

Step 3. *Variable count.* Increment q. If $q > n_k$ go to Step 6.

Step 4. *Reduced cost.* Increment i_k (mod n_k).

If $e^k_{i_k}$ is nonbasic, compute its reduced cost d, otherwise go to Step 3.

If d is non-profitable then go to Step 3,

otherwise increment r by 1.

If d is better than $d_{\max}$ then

d is the new value of $d_{\max}$ and the absolute index of $e^k_{i_k}$ is recorded in j.

Step 5. *Terminate cluster?* If the sufficient number (R) of improving vectors have been found exit from the cluster, i.e., if $r = R$ go to Step 6, otherwise go to Step 3.

Step 6. *Terminate pricing?* If the prescribed number (P) of clusters have been scanned and at least one candidate was found, terminate pricing, otherwise take next cluster, i.e., if $s \geq P$ and $d_{\max} > 0$ then go to Step 7, otherwise go to Step 1.

Step 7. *Evaluate termination.* If $d_{\max} > 0$ then variable with index j is a candidate to enter the basis, otherwise the simplex algorithm terminates. In the latter case if the current solution is feasible then it is optimal, if infeasible then the problem has no feasible solution.

Some comments on SIMPRI help identify its general usefulness.

- If d_j is explicitly maintained then 'compute reduced cost' can be replaced by 'retrieve reduced cost'.

- If normalized pricing is used it is usually done in the framework of full pricing. However, even in this case it is still possible to use some sort of a partial pricing. In either case, SIMPRI can be used with the evaluation criterion of the given normalized pricing method.

- SIMPRI is applicable to multiple pricing in a straightforward way. In this case, a pool of N_s best candidates is maintained, where N_s is a strategic parameter, with value usually $2 \leq N_s \leq 8$. Now the interesting issue is how to make the selection. It appears to be a good idea to keep candidates from different clusters with the understanding that they are relatively unrelated so that if of one of them enters the basis the profitability of the other selected candidates will not be damaged significantly.

- Pricing the nonbasic logical variables is computationally very simple and can easily be done whether and explicit d_j is maintained or not. Logicals can be considered as a single cluster, or, if there is information about rowwise subdivision they can be added to the appropriate clusters.

It remains to see if SIMPRI really does not overlook any special situation that can occur. The answer is formulated in the following proposition.

Proposition 5.1 *If the parameters are assigned meaningful values, i.e.,* $1 \leq K \leq n$, $1 \leq P \leq K$, *and* $1 \leq R \leq n$, *then the application of SIMPRI in the simplex method leads to a correct answer to the LP problem.*

Proof. It is enough to show that as long as there is at least one profitable candidate among the nonbasic variables, SIMPRI will find it, and SIMPRI will never fall into an endless loop.

First we show that if there is a single candidate it cannot be missed.

The algorithm will always start, because $K \geq 1$, therefore in the beginning relation "$s > K$" is false, so Step 2 is performed.

Since it is assumed that $R \geq 1$, there is always something to look for. A candidate in a cluster cannot be overlooked: Variables in a cluster are scanned sequentially in a circular order by the inner loop through Steps $3 - 4 - 5$. This loop can terminate in two ways: (i) through Step 5, if a sufficient number of profitable candidates were found, (ii) through Step 3, if the list is exhausted, i.e., all the elements in the given cluster have been checked.

If no candidate was found in the requested (P) number of clusters then Step 6 and Step 1 ensure that new cluster(s), taken in the circular order, will be scanned as long as there are unvisited clusters.

What would make SIMPRI try to loop infinitely? It may be thought that such case could happen if SIMPRI is given an unsolvable task, like to find more profitable candidates than presently available. A typical situation of this kind occurs when we are at an optimal solution (i.e., there is no profitable candidate at all). An endless loop cannot occur within a cluster, Step 3 takes care of it, since in a cluster the basic variables also increment the pointer. Similarly, clusters cannot be taken infinitely one after the other since Step 1 does not allow more than K clusters to be scanned, regardless of the index of the starting cluster. These two remarks ensure the proper termination of SIMPRI. $\square$

It is clear that SIMPRI creates a large flexibility in pricing. It enables us to tune the simplex algorithm for identified problem families by proper setting of the newly introduced parameters.

Of course, it may happen that a structure does not exist at all. Even in this case, if $n \gg m$ holds, some kind of a partial pricing with SIMPRI can help a lot in reducing the overall effort to solve the problem. Computational evidence shows [11] that SIMPRI with reasonably chosen clusters (not necessarily closely following any 'natural' clustering) can, in fact, be very efficient.

It is instructive to see how some known pricing schemes can be obtained as special cases of SIMPRI by appropriate setting of the three parameters.

1. If we define S as one cluster, requiring all nonbasic vectors to be scanned then we arrive at the generally known *full pricing*. This is achieved by the following setting of the parameters: $K = 1$, $P = 1$, and $R = n$. Clearly, here we do not expect that the number of improving vectors will be equal to n, but by this setting we can force the algorithm to scan all the nonbasic variables in every step. If the selection criterion is the unscaled max d_j then it reproduces the Dantzig pricing [4].

2. If we want to price one cluster fully per each iteration, taking one cluster after the other then we get the full *sectional pricing*. SIMPRI will work in this way for $K > 1$ if we set $P = 1$, and $R = n$.

3. Bland's rule can be obtained if we set $K = 1,\ P = 1,\ R = 1$ and restart SIMPRI in every iteration at Step 0.

4. If we have $K > 1$ and set $P = K$, $R = 1$, then we get the *one candidate from each subset* pricing strategy.

5. The frequently used simple *dynamic cyclic pricing* [1] is obtained if we set $K = 1$, $P = 1$, and R to some value $1 \leq R < |D|$, where $D = \{j \mid j$ nonbasic, $d_j > 0$ at a given step of the algorithm$\}$, i.e, the set of profitable candidates in the whole problem. D is usually determined after every reinversion of the basis.

6. Another example is Cunningham's LRC (Least Recently Considered) rule [3] for the network simplex method which is achieved by setting $K = 1$, $P = 1$, and $R = 1$.

On the basis of these examples we can say that SIMPRI is a certainly a generalization of several well known pricing schemes.

6 Concluding remarks

An obvious question arises. How do we know that the problem at hand has a structure that can be taken into account for pricing? Furthermore, if it exists, how can this information be passed over to the simplex solver?

In many cases the answer is quite straightforward as large scale problems are usually structured. They typically represent multi-period, multi-location, dynamic or stochastic models. Such problems are created and maintained by modelling systems. Therefore, the actual modelling system has all information needed for SIMPRI. The known structure can be encoded in the input (MPS) file prepared for the solver (MPS format being the *de facto* standard of representing LP problems). It can be placed either after the usual LP data or included in the 'column section' indicated by marker records, the same method which is applied to identify integer variables in an LP problem. It is very easy to adjust the input routines of the solvers to interpret this information and set up the structure needed for SIMPRI.

Another possibility is to visually inspect the distribution of nonzeros in the LP matrix. There are several tools available that can display the nonzero pattern. If K clusters of (nearly) equal size can be identified then cluster sizes (and thus boundaries) can well be approximated by taking $n_k = n/K$. In this way the benefits of any cluster-based pricing method can be enjoyed nearly in full.

It is important to emphasize that SIMPRI is just a framework. It can be extended, modified, customized or tuned in several different ways. To give some examples, we point to the following.

First of all, it is clear that SIMPRI can further be generalized by nesting the procedure when a higher hierarchy of structure exists, i.e., clusters within clusters.

It is an interesting variation when the selection criterion is a dynamically updated threshold value that can be different for each cluster. In this case, the first profitable candidate with a reduced cost greater than the threshold is selected and no further variables are scanned. Or, if multiple pricing is used, this idea can be applied to separate clusters to collect N_s candidates.

Clusters need not be visited in their 'natural' order. Sophisticated, dynamically updated ranking schemes among the clusters can be worked out but the main logic of SIMPRI still can be used.

Finally, SIMPRI can be used in the framework of the dual simplex method without any changes.

References

[1] M. Benichou, J.M. Gautier, G. Hentges, and G. Ribiere. The efficient solution of large-scale linear programming problems. *Mathematical Programming*, 13:280–322, 1977.

[2] R.G. Bland. New finite pivot rule for the simplex method. *Mathematics of Operations Research*, 2:103–107, 1977.

[3] W.H. Cunningham. Theoretical Properties of the Network Simplex Method. *Mathematics of Operations Research*, 4(2):196–208, May 1979.

[4] G.B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, 1963.

[5] J.J.H. Forrest and D. Goldfarb. Steepest edge simplex algorithms for linear programming. *Mathematical Programming*, 57(3):341–374, 1992.

[6] R. Fourer. Solving staircase linear programs by the simplex method, 2: Pricing. *Mathematical Programming*, 25:251–292, 1983.

[7] D. Goldfarb and J. Reid. A Practicable Steepest-Edge Simplex Algorithm. *Mathematical Programming*, 12:361–371, 1977.

[8] H.J. Greenberg. Pivot selection tactics. In H.J. Greenberg, editor, *Design and Implementation of Optimization Software*, NATO ASI, pages 143–174. Sijthoff and Nordhoff, 1978.

[9] P.M.J. Harris. Pivot Selection Method of the Devex LP Code. *Mathematical Programming*, 5:1–28, 1973.

[10] I. Maros. A structure exploiting pricing procedure for network linear programming. Research Report 18–91, RUTCOR, Rutgers University, NJ, USA, May 1991. 15 pages.

[11] I. Maros and G. Mitra. Investigating the Sparse Simplex Algorithm on a Distributed Memory Multiprocessor. *Parallel Computing*, 26:151–170, 2000.

[12] W. Orchard-Hays. *Advanced Linear-Programming Computing Techniques.* McGraw-Hill, New York, 1968.