A NETWORK PENALTY METHOD

by

A.B. Gamble*
A.R. Conn**
W.R. Pulleyblank*

Research Report CS-87-25

January 1987

*) Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada. Currently visiting the Institut für Ökonometrie und Operations Research, Universität Bonn, W. Germany.

**) Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

Financial support provided by the Natural Sciences and Engineering Research Council, Canada and Sonderforschungsbereich 303 (DFG), Institut für Ökonometrie und Operations Research, Universität Bonn, W. Germany.

A Network Penalty Method

by

A.B. Gamble
A.R. Conn
W.R. Pulleyblank

ABSTRACT

We consider the minimum cost network flow problem and describe how the linear penalty function methods of Conn and Bartels can be specialized to a combinatorial algorithm for this problem, the network penalty method. Computational experience indicates that this algorithm provides a significant improvement over the network simplex method. The algorithm can be proven finite using a modification of Cunningham's strongly feasible basis pivoting rule.

1. Introduction

Minimum Cost Network Flow problems are special instances of linear programs. The Minimum Cost Network Flow problem (NFP) is

$$\begin{aligned}&\text{minimize } cx\\&\text{subject to } Ax = b\\&0 \leq x \leq u,\end{aligned}$$

where A is the vertex–edge incidence matrix of a directed graph G, and b, c, and u are constant vectors. These problems have been successfully treated with combinatorial primal, dual, and primal–dual algorithms [7,8,9]. The standard Simplex method for linear programming can also be specialized to a combinatorial algorithm for solving (NFP). The resulting Network Simplex method has in practice compared favorably with the above mentioned combinatorial algorithms [10], and has often been the preferred method of solving Minimum Cost Network Flow problems. For a recent survey, see Grigoriadis [11].

Another approach which has been taken to linear programming is that of Penalty Function methods. These methods were developed originally for non–linear programming (Conn [2]) but have been specialized to algorithms for solving linear programs [1,3]. In particular, the algorithm presented by Bartels [1], is in structure quite similar to the Simplex method.

The basic idea is that the upper and lower bound constraints will no longer be required to be satisfied for each variable. Instead, if one of these bounds is violated, a penalty will be incurred in the objective function. This penalty will be based on some preset penalty parameter α, and will "encourage" the variable back towards its feasible range. This new penalty objective function $\Phi_\alpha(x)$ is minimized, subject to $Ax = b$, for some initial value of α. If the optimal solution to this new problem satisfies the upper and lower bound constraints for every variable, then it must be optimal to the original problem. Otherwise α is increased and we solve again. Computational results [1]have indicated that for reasonable initial values of α, this Penalty Function algorithm uses an average of 20 % fewer pivots than the Simplex method, on small, dense problems. Since the amount of work each method does during one pivot is comparable, this represents an improvement.

Our main objective is to show how these penalty function techniques can be applied to minimum cost network flow problems. This involves specializing the methods developed by Conn and Bartels to exploit the special structure present in (NFP). The result is an algorithm which can be viewed as a variant of the standard network simplex method. The main differences are the following:

i) When flow augmentations are performed, flow is not required to remain feasible on all edges.

ii) A consequence of i) is that the bases encountered during the course of the algorithm are not in general either primal or dual feasible. In fact, the algorithm can begin with any basis. This removes the necessity of adding artificial variables to the problem.

An important theoretical question regarding the Network Simplex method is that of finiteness [5,6]. Flow problems are, in general, highly degenerate, so even though instances of cycling are rare in practice, it is important to be able to prove an algorithm finite. Cunningham's method of strongly feasible bases [4] provides a simple pivoting rule which guarantees finiteness for the Network Simplex method. This rule restricts the set of bases which may be selected to the so called *strongly feasible* bases. One drawback of this method is that it must start with a strongly feasible basis, and such a basis may not be readily available. We present a modification of Cunningham's rule, *MCR* which also guarantees finiteness but has the added advantage that any starting basis may be used. This rule also restricts the set of bases which may be selected. In this case, however, the permissible subset is determined by the choice of starting basis. The new rule *MCR* is important for the Network Penalty algorithm, since this algorithm precludes the possiblilty of maintaining a strongly feasible basis. Another practical benefit of *MCR* is the following. If we have a Network Simplex pivoting rule which performs very well in practice but does not guarantee finiteness, then we can combine it with *MCR* to create a finite pivoting rule by simply applying *MCR* anytime we reach a sequence of pivots which we suspect to be cycling. Recently, strongly polynomial methods for the minimum cost network flow problem have been obtained by Tardos [14] and Orlin [13].

A final feature of our Network Penalty algorithm is that it does not always require that we minimize $\Phi_\alpha(x)$ for a fixed value of α. It is shown that it may be possible, before this happens, to recognize that the x minimizing $\Phi_\alpha(x)$ will not be feasible and hence α must be increased. Whenever this situation is recognized, we immediately increase α.

In the remainder of this section we introduce some of the elementary graph theoretic concepts and notation which will be used in this paper. In Section 2 we provide details of the network simplex method and Cunningham's pivoting rule. We also introduce the *MCR* pivoting rule. In Section 3 the general penalty method is reviewed and the added complication of degeneracy is considered. Section 4 concerns the early termination condition mentioned above. In Section 5 the network penalty algorithm is presented. Also included in this section is a proof, using *MCR*, that the network penalty algorithm is finite. Finally, the results of some computational experimentation are contained in Section 6.

A *directed graph* is a graph G such that for every edge $e \in E(G)$, one end of e is designated as the *head* of e (denoted by $h(e)$) and the other is designated as the *tail* (denoted by $t(e)$). We say that e is directed away from $t(e)$ and towards $h(e)$. The *incidence matrix* of a directed graph G is the matrix $A = [a_{i,j}]$ whose rows are indexed by the vertices of G, and whose columns are indexed by the edges of G, where

$$a_{i,j} = \begin{cases} 1 & \text{, if edge } j \text{ is directed towards vertex } i \\ -1 & \text{, edge } j \text{ is directed away from vertex } i \\ 0 & \text{, otherwise.} \end{cases}$$

If G is a directed graph, then the *underlying undirected graph* of G is the undirected graph having the same vertex set, edge set, and incidence relation as G. A *directed tree* is a directed graph whose underlying undirected graph is a tree. In a directed tree an edge e is said to be *directed towards* a vertex v if $h(e)$ is on the path connecting v and $t(e)$. Otherwise e is *directed away from* v. (Note that a directed tree is different from an arborescence, in which all arcs are directed away from some root node.) Given a directed graph G, a directed spanning subtree T, and an edge e not in T, the *fundamental cycle* $C(e)$ is the cycle in G consisting of $t(e)$ and e followed by the unique path in T from $h(e)$ to $t(e)$. An edge e_i on a cycle $C = (v_1, e_1, \dots, e_{k-1}, v_k)$ is a *forward* edge of C if $h(e_i) = v_{i+1}$. Otherwise it is a *backward* edge. Thus we have that e is always a forward edge of $C(e)$. The incidence vector of a cycle C is the vector p such that

$$p_j = \begin{cases} 1 & \text{, if } j \text{ is a forward edge of } C \\ -1 & \text{, if } j \text{ is a backward edge of } C \\ 0 & \text{, otherwise.} \end{cases}$$

If a cycle C has incidence vector p, then *reverse* C is the cycle having incidence vector $-p$. Given a cycle C and two edges $e, f \in C$, f is *directed with* e on C if they are either both forward edges of C or both backward edges. Otherwise f is *directed against* e on C. The graph G with the edge e deleted is denoted by $G - e$, and likewise $G + e$ denotes the graph G with the edge e added.

2. Network Simplex Method

All linear programming problems considered in this paper are of the following standard form:

$$\text{(LP)} \quad \begin{aligned} &\text{minimize } cx \\ &\text{subject to } Ax = b \\ &\qquad\qquad 0 \le x \le u. \end{aligned}$$

A minimum cost network flow problem (NFP) is an instance of (LP) where A is the incidence matrix of a directed graph G. For our purposes we will assume, without loss of generality, that G is connected. (NFP) has the following interpretation. We are given a *demand* b_v for each vertex v, and an *edge cost* c_e and *capacity* u_e for each edge of G. We wish to assign a number x_e to each edge e of G in such a way that,

i) for every $v \in V(G)$, the sum of numbers on edges directed towards v, minus the sum of numbers on edges directed away from v, equals the demand b_v at v,

ii) the number assigned to each edge is non–negative and not greater than the capacity of that edge,

iii) subject to i) and ii), we minimize the total cost, cx.

The dual problem (DFP) also has a special interpretation. It is the problem of assigning to each vertex, v, of G a number, y_v, and to each edge a number γ_e, in such a way that,

i) for every $e \in E(G)$, $y_{h(e)} - y_{t(e)} - \gamma_e \leq c_e$,

ii) for every $e \in E(G)$, γ_e is non–negative,

iii) subject to i) and ii), we maximize $yb - \gamma u$.

The sum of the rows of A is the zero vector, so unless $\sum_{v \in V(G)} b_v = 0$, the system $Ax = b$ will be inconsistent and will thus have no feasible solution. Therefore we assume that $\sum_{v \in V(G)} b_v = 0$. It follows that any one of the rows of $Ax = b$ is the negative of the sum of the others and thus can be deleted. We will arbitrarily choose a vertex and call it the *root*, denoted r. We will delete the constraint corresponding to r and equivalently, in the dual, set $y_r = 0$.

We now summarize the main features of the Network Simplex Method (see Kennington and Helgason [12]). A *basis* of an instance of (NFP) is a partition $B = (T, L, U)$ of the arcs of G such that T is the edge set of a spanning tree. The elements of T are called *basic* and those of $L \cup U$ are called *nonbasic*. Given a basis, the associated primal and dual basic solutions can be easily calculated in the following manner.

Primal Basic Solution:

Let $B = (E(T), L, U)$ be the given basis. Set $x_j = u_j$ for all $j \in U$. Adjust the vertex demands to account for these values. Let v be a vertex (other than r), which has degree 1 in T, and let e be the edge of T which is incident with v. Set x_e to satisfy the demand at v. Adjust the demand at the other vertex incident with e to account for this value of x_e. Delete e and v from T. Repeat this until r is the only vertex left in T. Set $x_j = 0$ for all edges $j \in L$.

Dual Basic Solution:

Set $y_r = 0$. Let v be a vertex which is adjacent to a vertex u in T where y_u has been set but y_v has not. Let e be the edge of T which joins them.

$$\text{Set } y_v = \begin{cases} y_u + c_e & \text{, if } e \text{ is directed towards } v \\ y_u - c_e & \text{, otherwise.} \end{cases}$$

Repeat until all node numbers have been set. For all $j \in (E(T) \cup L)$ set $\gamma_j = 0$. For all $j \in U$ set $\gamma_j = y_{h(j)} - y_{t(j)} + c_j$.

The Network Simplex Method is a specialization of the Simplex Method for solving Network Flow Problems. The method pivots from primal feasible basis to primal feasible basis attempting to find one which is also dual feasible. This algorithm proceeds as follows.

1. Start with a primal feasible basis, $B = (E(T), L, U)$.

2. Calculate the associated primal solution, x.

3. Calculate the associated dual solution, y.

4. For each edge j, compute the reduced cost $\bar{c}_j = c_j - y_{h(j)} + y_{t(j)}$.

   If $\bar{c}_j \geq 0$ for all $j \in L$, and $\bar{c}_j \leq 0$ for all $j \in U$, then B is dual feasible and thus optimal.

   Otherwise, find either $e \in L$ such that $\bar{c}_e < 0$, or $e \in U$ such that $\bar{c}_e > 0$.

5. Adding e to T will create exactly one cycle $C(e)$.

   $$\text{Let } C = \begin{cases} C(e) & \text{if } e \in L \\ \text{reverse } C(e) & \text{if } e \in U. \end{cases}$$
   $$\text{Let } \beta_j = \begin{cases} x_j & \text{for all backward edges } j \text{ of } C \\ u_j - x_j & \text{for all forward edges } j \text{ of } C. \end{cases}$$

   Let f be an edge in $C(e)$, such that β_f is minimum. (*)

   Let $\Theta = \beta_f$.

   Add Θ to the flow in all forward edges of C.

   Subtract Θ from the flow in all backward edges of C.

6. Set $T = T - f + e$.

   Delete e from L or U, whichever it was in.

   Set $L = L \cup \{f\}$, if f is a backward edge of C.

   Set $U = U \cup \{f\}$, if f is a forward edge of C.

   Go to Step 3.

Cunningham [4] presented a simple pivoting rule which ensured the Network Simplex Method would terminate in finite time. For this he introduced the concept of a strongly feasible basis.

Definition: A basis $B(T, L, U)$ is *strongly feasible* if for every edge $j \in T$,

$$x_j = 0 \text{ implies } j \text{ is directed away from } r \text{ in } T$$

and

$$x_j = u_j \text{ implies } j \text{ is directed towards } r \text{ in } T.$$

Now consider the following modifications to the Network Simplex algorithm:

(1) Initiate the algorithm with a strongly feasible basis.

(2) In Step 5, let F be the set of edges which are candidates to become non–basic. Let s be the first common vertex in the paths in T from $h(e)$ and $t(e)$ to r. Choose f to be the first member of F encountered in traversing $C(e)$ in the direction of e beginning at s.

Cunningham's theorem states that the Network Simplex algorithm with the above modifications is finite. However, this proof of finiteness requires that the initial basis be strongly feasible. For our application we require a rule which will guarantee finiteness regardless of whether the starting basis is strongly feasible. The following is a modification of Cunningham's rule, MCR, which does not require that we start with a strongly feasible basis. Therefore it has the practical advantage that we can begin applying the rule at any time we suspect cycling to be occurring. When the method makes a non–degenerate pivot, we can "turn off" the rule and thereby avoid the extra "overhead" it involves. This frees us from maintaining an anti–cycling mechanism throughout the algorithm, yet still guarantees finiteness. To implement MCR we will need to maintain a second vector of edge numbers $w = (w_j : j \in E(G))$.

Definition: A basis B will be *Semi–Strongly Feasible* (SSF) when for every edge j in the associated tree,

$$x_j = 0 \text{ and } w_j = 0 \text{ implies } j \text{ is directed away from } r$$

and

$$x_j = u_j \text{ and } w_j = 0 \text{ implies } j \text{ is directed towards } r.$$

Any time we wish to apply the finiteness procedure we make the following changes:

(1) Initially, for any edge j in the spanning tree such that $x_j = 0$ ($x_j = u_j$), and j is directed towards (away from) r, set $w_j = 1$ ($w_j = -1$). Set $w_j = 0$ for all other edges.

(2) In Step 5, replace statement (*), in which the leaving variable is chosen, with the following.

- Let F be the set of candidates to be non–basic, i.e. those e for which β_e is minimum.
- Let $\hat{\beta}_j = \begin{cases} w_j & \text{for all backward edges } j \text{ of } C \\ -w_j & \text{for all forward edges } j \text{ of } C \end{cases}$
- Let $\lambda = \min \{\hat{\beta}_j : j \in F\}$.
- Let $\hat{F} = \{j \in F : \hat{\beta}_j = \lambda\}$.

- Choose f from $\hat{F}$ as in CR.
- Add λ to w_j for all j directed with e on $C(e)$.
- Subtract λ from w_j for all j directed against e on $C(e)$.

The w's can be thought of as a secondary flow on the same network which satisfies a different set of node demands. Notice that if F contains only a single element, f will be chosen to be that element. The above rule is performed until a non–degenerate pivot is made, at which time the w's are discarded. Notice that in the special case where our starting basis is strongly feasible, the w's will all be initially set to 0, and this method will duplicate Cunningham's rule.

Theorem 1 *The Network Simplex algorithm with the above modifications is finite.*

Proof. The proof of this theorem is in two parts. Its structure is the same as the proof of Cunningham's theorem. First we show that the MCR pivoting rule causes the Network Simplex algorithm to encounter only semi–strongly feasible bases.

Let B^0 be a semi–strongly feasible basis. Let B^1 be a basis obtained from B^0 by MCR. Let T^0 and T^1 be the associated spanning trees and let x^0, y^0 and x^1, y^1 be the associated primal and dual basic solutions.

Note that edges of T^1 which are not in $C(e)$ are directed towards r in T^1 if and only if they are directed towards r in T^0. Therefore we need only consider edges of $C(e)$. There are two cases.

1. $\Theta > 0$ or $\lambda > 0$.

In this case the edges $j \in C(e) \cap T^1$ such that $w_j^1 = 0$ and $x_j^1 = 0$, or $w_j^1 = 0$ and $x_j^1 = u_j$, are precisely the edges of $\hat{F} - \{f\}$. MCR ensures that each of these is directed appropriately, either towards or away from r in T^1.

2. $\Theta = 0$ and $\lambda = 0$.

In this case $x^1 = x^0$ and $w^1 = w^0$. Let $x_e^1 = 0$ ($x_e^1 = u_e$). Since B^0 is semi–strongly feasible, every element of $\hat{F}$ must lie on the path between $h(e)$ and $s(t(e)$ and $s)$ in T^0. Therefore e will be directed away from (towards) r in T^1. Also, as in case 1, the members of $\hat{F} - \{f\}$ will be directed appropriately.

In both cases B^1 is semi–strongly feasible.

We now show that MCR will never permit a basis to be encountered more than once during a sequence of degenerate pivots. That is, we need only concern ourselves with pivots for which $\Theta = 0$. Again there are two cases.

1. $\lambda > 0$.

In this case the pivot may be thought of as a non–degenerate pivot with respect to the flow w. Therefore the function cw will be strictly decreasing.

2. $\lambda = 0$.

Let R be the set of vertices in the component of $T^0 - f$ containing r. Notice that by definition,

$$\text{if } i \in R, \text{ then } y_i^1 = y_i^0,$$
$$\text{if } i \notin R, \text{ then } y_i^1 = \begin{cases} y_i^0 + c_e - y_{h(e)}^0 + y_{t(e)}^0 < y_i^0 & , \text{if } e \in L \\ y_i^0 - c_e + y_{h(e)}^0 - y_{t(e)}^0 < y_i^0 & , \text{if } e \in U. \end{cases}$$

Therefore $\sum_{i \in V(G)} y_i^1 < \sum_{i \in V(G)} y_i^0$. That is, the sum of the components of y strictly decreases with each degenerate pivot. Both the w's and the dual variables are uniquely determined by the basis, so no basis may be repeated and the theorem follows.

□

3. Penalty Function Methods

Consider an instance of (LP), our standard linear programming problem. It is known that an optimal solution to (LP) may be found by minimizing an unconstrained piecewise linear penalty function. Instead of explicitly requiring that the constraints be satisfied, a penalty is incurred in the objective function for each violated constraint. Conn [3] introduced an algorithm to minimize one such penalty function. His approach proceeds by finding a direction of decrease and incrementing along that direction until the penalty function is no longer decreasing. If the solution thus obtained is feasible for (LP), then it must be optimal for (LP). Otherwise the penalty for violated constraints is increased and the process continued.

We may, of course, choose any subset of the constraints and require that they be satisfied explicitly rather than representing them in the objective function. One possible approach is to impose the equality constraints ($Ax = b$) explicitly, while including penalties in the objective function for violated upper and lower bound constraints. In this case the penalty function considered by Conn will take the following form:

$$\Phi_\alpha(x) = cx - \alpha \sum_{j=1}^{n} \min(x_j, 0) - \alpha \sum_{j=1}^{n} \min(u_j - x_j, 0) \ ,$$

where $\alpha > 0$ is some real constant. The penalty function problem we consider is

$$\begin{aligned}\text{(PP)}\ \ \text{Minimize}\ & \Phi_\alpha(x)\\ \text{subject to}\ \ & Ax = b.\end{aligned}$$

This problem obviously is related to (LP), but (PP) and (LP) generally do not have the same solutions. For example, if α is very small, (PP) simply minimizes cx subject to $Ax = b$ effectively ignoring the upper and lower bound constraints. Hence it is possible for (PP) to have an optimal solution which is not even feasible for (LP). However, if an optimal solution of (PP) satisfies $0 \le x \le u$, then it is also an optimal soution of (LP). Moreover, if (LP) has a feasible optimal solution, then there exists $\overline{\alpha}$ such that $\alpha \ge \overline{\alpha}$ implies any solution of (PP) will be optimal if and only if it is an optimal solution of (LP) [1,3]. One well known bound on $\overline{\alpha}$ is that it is no greater than the largest (in magnitude) dual variable in an optimal solution. Notice that the objective function $\Phi_\alpha(x)$ can be rewritten,

$$\Phi_\alpha(x) = \sum_{0 \le x_j \le u_j} c_j x_j + \sum_{x_j < 0} (c_j - \alpha) x_j + \sum_{x_j > u_j} (c_j + \alpha) x_j - \alpha \sum_{x_j > u_j} u_j \ .$$

Thus once the current x is known, the current coefficients $c_j^{'}$ of the objective function can easily be obtained. That is,

$$c_j^{'} = \begin{cases} c_j & \text{if } 0 \le x_j \le u_j \\ c_j - \alpha & \text{if } x_j < 0 \\ c_j + \alpha & \text{if } x_j > u_j \ . \end{cases}$$

Bartels [1] presented an algorithm for solving (PP) which is a variation on Conn's approach. Unlike Conn's method, this algorithm pivots from basis to basis in a simplex–like manner, although it requires neither primal nor dual feasibility to be maintained. However, just as in the case of the simplex method, for every solution constructed by the algorithm, each nonbasic variable will equal either its upper or lower bound. This means that infeasibilities can occur only on basic variables. Bartels' algorithm makes use of the assumption that the linear program is non–degenerate. That is, no basic variable can have its value equal to either of its bounds. This assumption allows the assignment of a single reduced cost $\overline{c}_j$ to every nonbasic variable x_j, so that $\overline{c}_j$ reflects the per unit effect on the objective function of either increasing or decreasing the value of x_j. In particular, if x_j is at lower bound, then $\overline{c}_j$ is the per unit effect of increasing x_j and $-\overline{c}_j + \alpha$ is the per unit effect of decreasing x_j. Similarly, if x_j is at upper bound, then $-\overline{c}_j$ is the per unit effect of decreasing x_j, and $\overline{c}_j + \alpha$ is the per unit effect of increasing x_j.

As previously mentioned, minimum cost network flow problems are in general highly degenerate. This means that some basic variables may be at either upper or lower bound and will thus have different costs, depending on whether they are increased or decreased. This, in turn, means that the reduced cost of a nonbasic variable can no longer be represented easily as one number. Instead, for each nonbasic variable, two seperate reduced costs must be calculated. To calculate these numbers one must know the number of basic variables which will violate their bounds immediately upon a change in the nonbasic variable. This makes the degenerate case substantially more complicated. Care must be taken or we may be left with an algorithm which could terminate with a suboptimal solution. For an example of this, see Figure 1 which is presented within the framework of network flow problems.

$b_v = 0$ for all vertices v
$u_e = 1$ for all edges e
$c_e = 1$ for all edges e
$T = \{4,5,6,7\}$
$U = \{1,2,3\}$
$L = \emptyset$

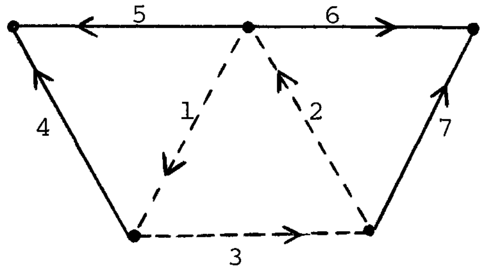

Figure 1

The basic solution x' associated with the basis (T, L, U) assigns a value of one to arcs 1,2 and 3, and a value of zero to all others. The reduced costs associated with increasing (c_j^+) and decreasing (c_j^-) each nonbasic arc, j, are listed below in terms of the penalty parameter α.

$$\begin{aligned} c_1^+ &= 1 + 2\alpha & c_1^- &= -1 + \alpha \\ c_2^+ &= 1 + 2\alpha & c_2^- &= -1 + \alpha \\ c_3^+ &= 1 + 3\alpha & c_3^- &= -1 + 2\alpha \end{aligned}$$

Thus, for $\alpha > 1$, all reduced costs will be positive and the algorithm will terminate with the solution x'. It is clear, however, that when $\alpha > 1$, the zero flow is the unique optimum. The reason that the algorithm fails on this example is that all non–degenerate pivots require the violation of a bound and thus, for large α, these pivots do not look profitable.

A solution to this problem is to give each basic variable a single cost. If the value is feasible (even if it is at bound) this cost is just the original. If it is infeasible, the cost will include the penalty parameter α. This means that if a basic variable at its bound is to become infeasible, the situation is treated as a bound which is reached immediately. Thus the method may make degenerate pivots. If this method is used on the problem in Figure 1, then arcs 1,2 and 3 (in that order) will be pivoted into the basis, the first two pivots being degenerate. Thus we will obtain the optimal solution.

When applied to (NFP), the above approach also has a computational advantage. Since each basic (tree) arc has a single cost, dual variables can be calculated in the normal fashion and these can be used to calculate the reduced costs of the nonbasic arcs.

This approach, however, creates a problem with cycling in that infinite sequences of degenerate pivots become not only possible, but common. An example of this situation is given in Figure 2.

$b_v = 0$ for all vertices v
$u_e = 1$ for all edges e
$c_1 = c_2 = 2$ $c_3 = 1$
$\alpha = 2$
$T = \{2,3\}$
$L = \{1\}$, $U = \emptyset$
$x_e = 0$ $\forall$ edges e

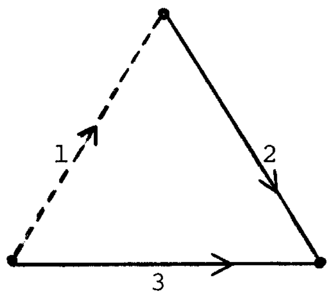

Figure 2

The reduced cost for lowering x_1 is -1, so arc 1 enters the basis and arc 2 leaves. This is a degenerate pivot. Now the reduced cost for lowering x_2 is -1, so arc 2 enters the basis and arc 1 leaves. Thus we have cycled back to the original basis. In general, this situation can arise any time the reduced cost for moving a non basic variable away from its feasible region is negative but greater than $-\alpha$. If the associated fundamental cycle contains an arc at bound, then that arc will leave the basis but will immediately become a candidate for re–entry. More complicated situations, involving many variables, are also common.

The reason that cycling is so likely in this algorithm is that we allow non-basic variables to move away from their feasible regions. However, the main result of the next section is the following: In view of our overall goal of solving (NFP) rather than (PP), those pivots which move nonbasic variables away from the feasible region are not required. If these are the only pivots available, then either the current solution is optimal for (NFP), or the penalty parameter α is too small and should be increased.

4. Early Termination Condition

One possible variant of the Penalty Function algorithm would permit the flow on a non–basic edge to move only into its feasible region. That is, non–basic edges at their upper bound could only be decreased and non-basic edges at thier lower bound could only be increased. Thus the method would continue pivoting until it encountered a basis which was dual feasible with respect to the penalty objective function coefficients c' (as defined in § 3). This would restrict the set of possible pivots and hence the method could terminate without actually minimizing $\Phi_\alpha(x)$. For example see Figure 3.

$\alpha = 1$
$c_1 = c_2 = 0$
$c_3 = 2$
$u_1 = u_2 = u_3 = 1$
$b_v = 0$ for all vertices v
$E(T) = \{1,2\}$
$L = \{3\}$, $U = \emptyset$

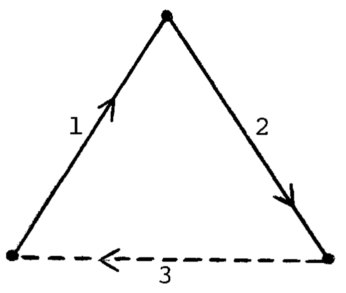

Figure 3

In Figure 3, consider the flow x^0, which is zero in every edge, so that $\Phi_\alpha(x^0) = 0$. Increasing the flow on edge f would increase the value of $\Phi_\alpha(x)$, so our Early Termination form of the Penalty Function algorithm would terminate with this flow. However, if we let $x^1_d = x^1_e = 1$ and $x^1_f = -1$, then all vertex demands are satisfied by x^1, and $\Phi_\alpha(x^1) = -1$, so x^0 obviously does not minimize $\Phi_\alpha(x)$. Recall, though, that our overall objective is to solve (NFP). If the dual feasible basis we encounter happens to be primal feasible also, then it must be optimal for (NFP). If not, then we can continue to minimize $\Phi_\alpha(x)$, or we have the option (as we do at any stage in the algorithm) of immediately increasing α. Bartels takes the latter option in his algorithm, interpreting the existance of a dual feasible, primal infeasible basis as an indication that the minimizers of $\Phi_\alpha(x)$ are likely not primal feasible. In the remainder of this section we will show that the existence of such a basis is in fact a guarantee that the minimizers of $\Phi_\alpha(x)$ are not primal feasible. For the remainder of this section, we call a cycle, C, *feasible* with respect to a basis B, if for every edge e on C, $e \in L$ implies that e is a forward edge of C and $e \in U$, implies that e is a backward edge of C.

Consider a basis B, with primal and dual solutions x and (y, γ), which is dual feasible with respect to c'. This means there are no negative cost feasible fundamental cycles. It is however possible for there to be negative cost fundamental cycles which are not feasible, in which case x may not be a minimizer for (PP). Now, if x satisfies $0 \leq x \leq u$, then $c' = c$, and thus B is an optimal basis for (NFP). Theorem 2 takes care of the case where x does not satisfy $0 \leq x \leq u$. It allows us to state that in this situation α is not large enough since no optimal solution to this instance of (PP) can be a feasible solution to our original problem (NFP). We first state some simple preliminary results.

(1) Let G be a directed graph with incidence matrix A and let C be a cycle in G. If C has incidence vector p, then $Ap = 0$.

(2) Let x satisfy $Ax = 0$, where A is the incidence matrix of a directed graph G. Then, there exist cycles C_i with incidence vectors p_i for $i = 1, \ldots, m$ such that

$$x = \sum_{i=1}^{m} \omega_i p_i \ ,$$

where $\omega_i > 0$ is a real number for $i = 1, \ldots, m$. Furthermore, for each of the cycles all backward edges j satisfy $x_j < 0$ and all forward edges j satisfy $x_j > 0$.

(3) Let G be the directed graph of a Min Cost Network Flow problem and let $B = (T, L, U)$ be a basis of the problem. If C is a feasible cycle in G with incidence vector p, then, $p = \sum_{e \in L \cap C} p^e - \sum_{e \in U \cap C} p^e$, where p^e is the incidence vector of $C(e)$.

Statements (1) and (2) are immediate. Statement (3) follows from the fact that the set of fundamental cycles in any graph (with spanning tree T) spans the cycle space.

Theorem 2 *Let B^1 be a dual feasible basis whose basic solution x^1 does not satisfy $0 \leq x^1 \leq u$. Then there does not exist any basis B^2 having basic solution x^2 such that $0 \leq x^2 \leq u$ and $\Phi_\alpha(x^2) < \Phi_\alpha(x^1)$.*

Proof. Assume that there exist such bases B^1 and B^2 with associated primal solutions x^1, and x^2. Consider a new flow x defined by

$$x_j = x_j^2 - x_j^1 \ \text{ for every edge } \ j.$$

We have $Ax = 0$ and therefore, by statement (2), $x = \sum_{i=1}^{m} w_i p_i$, where the p_i's are the incidence vectors of cycles satisfying the conditions of the statement. Now define c^1 by,

$$c_j^1 = \begin{cases} c_j & \text{if } j : 0 \leq x_j^1 \leq u_j \\ c_j - \alpha & \text{if } j : x_j^1 > 0 \\ c_j + \alpha & \text{if } j : x_j^1 < u_j . \end{cases}$$

Thus c^1 is the vector of cost coefficients for x^1 in the objective function $\Phi_\alpha(x)$. We now show that $c^1x < 0$. We have

$$\begin{aligned} c^1x &= \sum_{0\le x_j^1\le u_j} c_jx_j + \sum_{x_j^1<0}(c_j-\alpha)x_j + \sum_{x_j^1>u_j}(c_j+\alpha)x_j. \\ &= cx - \alpha\sum_{x_j^1<0} x_j + \alpha\sum_{x_j^1>u_j} x_j \end{aligned}$$

But we know that

$$\Phi_\alpha(x^2) - \Phi_\alpha(x^1) = cx + \alpha\sum_{x_j^1<0} x_j^1 - \alpha\sum_{x_j^1>u_j}(x_j^1-u_j), \quad \text{since} \quad 0 \le x^2 \le u.$$

Therefore,

$$\begin{aligned} c^1x &= \Phi_\alpha(x^2) - \Phi_\alpha(x^1) - \alpha\sum_{x_j^1<0}(x_j+x_j^1) + \alpha\sum_{x_j^1>u_j}(x_j+x_j^1-u_j) \\ &= \Phi_\alpha(x^2) - \Phi_\alpha(x^1) - \alpha\sum_{x_j^1<0} x_j^2 + \alpha\sum_{x_j^1>u_j}(x_j^2-u_j) \\ &< 0, \end{aligned}$$

since $\Phi_\alpha(x^2) < \Phi_\alpha(x^1)$, and $0 \le x^2 \le u$.

Therefore, since x is a sum of weighted cycles, at least one of those cycles C must be a negative cost cycle with respect to the edge costs c^1. Also, from statement (2), C will have the property that $x_j = x_j^2 - x_j^1 < 0$ for every backward edge, and $x_j = x_j^2 - x_j^1 > 0$ for every forward edge. Therefore every backward edge j of C will satisfy $x_j^1 > 0$, and every forward edge j will satisfy $x_j^1 < u_j$. Thus C is a feasible cycle with respect to the flow x^1 and by statement (3),

$$p = \sum_{e\in L\cap C} p^e - \sum_{e\in U\cap C} p^e,$$

where p is the incidence vector of C and p^e is the incidence vector of $C(e)$.

Therefore, either there exists $e \in L$ such that $C(e)$ is a negative cost feasible fundamental cycle with respect to x^1 and c^1, or there exists $e \in U$ such that reverse $C(e)$ is such a cycle. But this contradicts the fact that B^1 is dual feasible, and the Theorem follows.

□

This theorem tells us that it is not always necessary to actually minimize $\Phi_\alpha(x)$. It is sufficient to solve the following problem:

(PP*) Given an instance of (PP), find a basis which is dual feasible with respect to the coefficients c' of $\Phi_\alpha(x)$.

We note here that analogous statements to (1), (2) and (3) are true for the general linear programming case and thus Theorem 2 holds in this more general setting. Therefore, when using the penalty method for general linear programs it is also sufficient to solve (PP*) rather than (PP).

5. Network Penalty Algorithm

We now specialize Bartels' algorithm to the case of Network Flow problems. At any given time we will have a current basis B, an associated spanning tree T, and corresponding primal solution x. Once the current x is known, the current coefficients c' of the objective function can be easily obtained. Using these values we can find the dual solution y associated with B as described in §2.

There are two basic steps in the algorithm. First we must find a suitable cycle on which to increment the flow, and secondly we must decide how much the flow should be incremented. As indicated above, we will consider only fundamental cycles for possible incrementation. This is merely a matter of preference. While it may be more in keeping with the philosophy of Conn's algorithm to consider all cycles, we will chose not to do so in order to maintain the simplex–like structure of the algorithm. In a future paper we will explore the effects of removing this restriction.

The choice of incrementing cycle is made exactly as in the Network Simplex method. Since we are attempting to solve (PP*), we first check whether the current basis is dual feasible. If it is not, then there is either some $e \in L$ such that $\bar{c}'_e = c'_e - y_{h(e)} + y_{t(e)} < 0$, or some $e \in U$ such that $\bar{c}'_e > 0$. Let $C = C(e)$ if $e \in L$, and let $C =$ reverse $C(e)$ if $e \in U$.

Now let us consider how the step size Θ, is chosen. Let e be an edge not in T such that, $\bar{c}'_e < 0$ and $e \in L$ ($\bar{c}'_e > 0$ and $e \in U$). Adding e to T creates $C(e)$, and we increment the flow around $C(e)$ by adding $\Theta p(-\Theta p)$ to the flow for some $\Theta \geq 0$. In the Network Simplex method Θ is set to be the largest value satisfying $0 \leq (x + \Theta p) \leq u$. In the penalty algorithm, though, we do not necessarily stop increasing Θ at this point. We instead increment until we are no longer improving the value of $\Phi_\alpha(x)$. This will occur once $\bar{c}'_e$ is no longer less than (greater than) zero. Let f be an edge on $C(e)$ whose flow, as we increase Θ, reaches either $x_f = 0$ or $x_f = u_f$. There are two possible cases:

i) x_f is being decreased.

In this case f must be directed against (with) e on $C(e)$. If we continue to increment the flow on $C(e)$ past this bound, the coefficient of x_f in $\Phi_\alpha(x)$ will be decreased by α. To obtain equality at the dual constraint for edge f, we must add α to $y_{t(f)}$ and likewise for every vertex around the cycle up to and including $y_{t(e)}$ $(y_{h(e)})$. This has the effect of adding $\alpha(-\alpha)$ to $\bar{c}'_e$. If this value is still less than (greater than) zero, then we can continue to improve $\Phi_\alpha(x)$ by moving past this bound. If not, then we should stop increasing Θ..

ii) x_f is being increased.

In this case f must be directed with (against) e on $C(e)$. If we continue to increment the flow on $C(e)$ past this bound, the coefficient of x_f in $\Phi_\alpha(x)$ will be increased by α. To obtain equality at the dual constraint for edge f we must add $-\alpha(\alpha)$ to $y_{t(f)}$ and likewise for every vertex around the cycle up to and including $y_{h(e)}$ $(y_{t(e)})$. The effect of this is the same as in case i).

In both of these cases α is added to (subtracted from) $\bar{c}'_e$ each time we pass a bound. Therefore it remains worthwhile to increment the flow until $\left\lceil \frac{|\bar{c}'_e|}{\alpha} \right\rceil$ bounds have been reached. This is the point where we are no longer improving $\Phi_\alpha(x)$, and thus it is the point to which we should increment the flow. Notice that this method of determining Θ does not depend on the equality constraints $(Ax = b)$, and thus is applicable to the general linear programming penalty algorithm.

We now present the **Network Penalty Algorithm.**

1. Start with any basis, $B = (T, L, U)$.
2. Calculate the primal solution, x.
3. Calculate the dual solution y using the penalty function coefficients c' (§3.1).
4. For each edge j, let $\bar{c}'_j = c'_k - y_{h(j)} + y_{t(j)}$.

   Find either $e \in L$ such that $\bar{c}'_e < 0$

   or $e \in U$ such that $\bar{c}'_e > 0$.

   If no such e exists, then B is dual feasible with respect to c'. Stop.

   Otherwise, let $N = \left\lceil \frac{|\bar{C}'_e|}{\alpha} \right\rceil$.

5. Adding e to T creates the cycle $C(e)$.

   Let $C = \begin{cases} C(e) & \text{if } e \in L \\ \text{reverse } C(e) & \text{if } e \in U \end{cases}$

   Let $\beta_j^L = \begin{cases} x_j & \text{for all backward edges of } C \\ -x_j & \text{for all forward edges of } C\backslash e \end{cases}$

   Let $\beta_j^U = \begin{cases} x_j - u_j & \text{for all backward edges of } C\backslash e \\ u_j - x_j & \text{for all forward edges of } C \end{cases}$

   Order the non–negative β's such that $\beta^i_{f_1} \leq \ldots \leq \beta^i_{f_t}$ where $i \in \{L, U\}$.

   If $t < N$, then the objective value will continue to improve even after the last bound has been passed, and thus the problem is unbounded. Stop.

   Let $f = f_N$, and set $\Theta = x_f$.

   Add Θ to the flow in all edges directed with e in $C(e)$.

   Subtract Θ from the flow in all edges directed against e in $C(e)$.

6. Set $T = (T - f) + e$

   Go to Step 3.

This algorithm can now be used as part of an overall algorithm to solve (NFP). Start with some value for α, and run the Network Penalty algorithm. If it gives us a basis which is primal and dual feasible, then we have found an optimal solution to (NFP). Otherwise, increase α and continue from step 3. As is the case with the general penalty algorithm, α need not be increased beyond the magnitude of the largest dual variable in an optimal solution. In the case of (NFP), a particularly simple bound on this value is provided by the number of vertices times the maximum edge cost. Thus if after α reaches this bound, the optimal basis for (PP*) is still not primal feasible, then we know that no primal feasible solution for that particular instance of (NFP) exists.

It is evident that the Network Penalty algorithm presented above is very similar to the Network Simplex method as presented in §2. Converting existing computer code from one to the other is very easy since all the same data structures can be used.

We now consider the question of finiteness of the penalty algorithm. The overall algorithm will be finite if the Network Penalty algorithm to solve (PP*) is finite. As with the Network Simplex method, the Network Penalty algorithm moves from basis to basis. Thus if we can show that no basis is ever encountered more than once, we have proven the algorithm to be finite.

Consider the following pivoting rule.

- If there is more than one candidate to become non–basic, make the choice arbitrarily if the pivot is non–degenerate.

- During a sequence of degenerate pivots, use the MCR pivoting rule. For the purposes of the rule, treat all edges j such that $x_j < 0$ as if at lower bound and all edges j such that $x_j > u_j$ as if at upper bound.

Thus once it is determined that a pivot will be degenerate, the ω vector of MCR must be determined and maintained until such time as a non–degenerate pivot occurs.

Theorem 3 *The Network Penalty algorithm with the above pivoting rule is finite.*

Proof. For any non–degenerate pivot the value of the penalty objective function $\Phi_\alpha(x)$ is decreased and during a degenerate pivot its value does not change. Since $\Phi_\alpha(x)$ is determined by the basis, no sequence of pivots which cycles can include any non–degenerate pivots. But during a sequence of degenerate pivots, MCR will ensure that no basis is encountered more than once.

□

6. Computational Results

A computer program was written to implement the Network Penalty algorithm, as presented in §5, and to compare it with the Network Simplex method. When using the Network Penalty method, one immediately encounters the following two questions: What value should be initially assigned to α and by how much should α be increased if the algorithm does not terminate with a solution satisfying $0 \leq x \leq u$? Unfortunately, very little theory has been developed in this area, so the values of α were chosen on the basis of experimentation.

If α is very small, i.e. a very small penalty is assigned to violated bounds, then the resulting subproblem will almost invariably be unbounded. Even though this may be detected very early, often even on the first pivot, it seems that the solution of such a subproblem is of very little use in solving (NFP). It appeared that, if α was large enough to avoid this trivial situation, the solution of each subproblem required a significant number of pivots. Consequently, if several successive values of α were needed, the total number of pivots was large. Thus we decided to increase α by a multiple of 10 after an infeasible termination.

To study the problem of selecting an initial value for α, we arbitrarily chose a constant problem size, generated several random problems of that size, and tested the algorithm with various starting values of α. The networks were generated as follows. After the desired number of vertices (n), and edges (m) were chosen, a random spanning tree was formed to ensure the graph was connected. Since there are $n(n-1)$ edges in the complete directed graph on n vertices, this left $(n-1)^2$ possible places for the remaining $m-(n-1)$ edges. Each of these was given probability $\frac{m-n+1}{(n-1)^2}$ of being included in the network. Thus, m was the expected number of edges in the network. The spanning tree formed above was used as an initial basis by both methods with the flow in all remaining edges being set to zero. The Simplex method added artificial edges where necessary.

We investigated the effects of different α values on problems of size 100 nodes and 500 arcs. Arc costs were chosen as uniform random integers from the range –100 to 100. Arc capacities were chosen similarly from the range 1 to 200 and node demands from the range –30 to 30.

The pivot rule used by the Simplex method was to choose the entering variable as the one with the largest dual violation (as measured by the reduced cost) and to choose the leaving variable, in case of ties, arbitrarily. The same rule was used by the Penalty method.

Five problems with the above parameters were randomly generated and tested. The results of these tests are shown in table 1. For each of the problems there are three columns. The first (R1), is the number of iterations used in the first run of the penalty method, the second (R2) is the number of iterations needed in the second run (after increasing α) and the third is the total. For each problem the best total obtained is indicated with an asterisk. A (u) appearing in column R1 indicates that the initial problem had an unbounded solution. The table shows results for initial values of α ranging from 40 to 300. In fact values up to 500 were tested, but the results were generally about the same as for $\alpha = 300$. The results for $\alpha = 10,000$ are included to demonstrate the performance of the Penalty method when a very large value of α is used. The final row of the table shows the number of iterations required by the Simplex method for each of the problems. It is interesting to note that, for large α, once a primal feasible basis is obtained, no further primal violations will be incurred. Therefore, if the algorithms had been initiated with primal feasible bases, the number of iterations required when $\alpha = 10,000$ would have been equal to the number required by simplex.

From the table we can see that a value of $\alpha \simeq 200$ appears be the best. More striking than this, however, is the fact that the best value of α is in all cases approximately equal to the smallest value of α for which the problem is solved in one run. This suggests that it is best to solve (or very nearly solve) the original (NFP) problem with the first run and subject to this make α as small as possible. In problems of our "standard" size this means setting α to be approximately 200.

Next, several of the problem parameters were varied. Varying the arc cost range has a direct effect on what the best choice for α is. That is, if the cost range is multiplied by some constant k, then α should also be multiplied by k.

Varying the range of arc capacities and node demands had a somewhat surprising effect. While the best starting value of α remained unchanged, the overall performance of the Network Penalty algorithm was greatly affected. The penalty method performed much better on problems with large range of arc capacities and node demands. On these problems, about a 20 % improvement over the Simplex method (in terms of number of simplex pivots) was observed. That is, the Simplex method took about 25 % more pivots to solve the problems than did the Penalty method. The original five test problems were used for these tests, varying only the range of arc capacities and node demands. For all tests a starting value of $\alpha = 200$ was used. For each problem the percentage improvement of the Penalty method over the Simplex method (in terms of number of simplex pivots) was calculated and the highs, lows and medians of these values are recorded in table 2.

As the range of possible arc capacity and node demand values was narrowed, the percentage improvement of the Penalty method over the Simplex method was sharply reduced (see table 2). Moreover, the Penalty method did not perform as well on assignment problems, (where arc capacities are constant and node demands have only two possible values). In testing on random assignment problems, the Penalty method averaged only a 5 % improvement over the Simplex method. One possible reason for this decrease in performance is the following. While augmenting the flow of a cycle in such a problem, when the point is reached where a primal violation must be incurred (or the augmentation stopped), it is likely that several primal violations will be incurred at the same point. Thus there is a strong disincentive for the Penalty method to commit primal violations. Thus it will, to a certain degree, mimic the Simplex method.

Finally, the effect of increasing the problem size was explored. Interestingly, increasing the problem size had very little effect on the best starting value of α. An initial value between 200 and 250 was found to be best for all problem sizes tested. As can be seen in table 3, the percentage improvement of the Penalty method over the Simplex method increased steadily with the problem size. On the largest problems (700 nodes, 3500 arcs) an average of approximately 33 % improvement was observed, that is the Simplex method took 50 % more pivots to complete the problems than did the Penalty method.

Comparison of initial α values on problems of 100 nodes and 500 arcs.

Problem number	1			2			3		
α	R1	R2	TOT	R1	R2	TOT	R1	R2	TOT
40	8(u)	435	443	1(u)	369	370	1(u)	437	438
60	8(u)	451	459	1(u)	370	371	19(u)	445	464
80	117(u)	338	445	184(u)	316	500	137(u)	300	437
100	341	109	450	316	124	440	360	82	442
120	362	64	426	314	145	459	363	34	397
140	344	18	362*	339	107	446	388	5	393
160	367	19	386	343	17	360	380	0	380*
180	389	0	389	333	21	354	423	0	423
200	378	0	378	354	2	356	410	0	410
220	420	0	420	334	0	334*	418	0	418
240	424	0	424	335	0	335	388	0	388
260	378	0	378	347	0	347	407	0	407
280	433	0	433	352	0	352	399	0	399
300	418	0	418	360	0	360	428	0	428
10000	429	0	429	391	0	391	424	0	424
Simplex			537			439			457

Table 1

(u) : Unbounded Optimum
* : Fewest Iterations

	4			5		
α	R1	R2	TOT	R1	R2	TOT
40	8(u)	405	413	1(u)	391	392
60	134(u)	318	452	4(u)	428	432
80	286	246	532	282	277	559
100	351	104	455	331	123	454
120	328	42	370	344	71	415
140	386	12	398	372	72	444
160	402	3	405	365	11	376
180	378	12	390	369	10	379
200	355	0	355*	360	9	369*
220	389	0	389	378	0	378
240	412	0	412	423	9	432
260	403	0	403	390	0	390
280	415	0	415	400	0	400
300	387	0	387	384	0	384
10000	433	0	433	456	0	456
Simplex			465			380

Table 1 (continued)

(u) : Unbounded Optimum
* : Fewest Iterations

Effect of Varying b and u Ranges

Problem parameters	% improvement High	Low	Median
$b \in [-3,3]$ $u \in [1,20]$	22.3%	0.7%	13.1%
$b \in [-30,30]$ $u \in [1,200]$	29.6%	10.3%	22.3%
$b \in [-300,300]$ $u \in [1,2000]$	23.0%	17.8%	21.5%
$b \in [-3000,3000]$ $u \in [1,20000]$	22.6%	9.9%	18.2%

Table 2

Effect of Increasing Problem Size

$n = 100$ $e = 500$ % difference

Problem number	1	2	3	4	5	High	Low	Med.
Simplex	493	417	492	467	495			
Penalty, $\alpha = 200$	393	341	401	375	402	20.3	18.2	18.8
Penalty, $\alpha = 250$	410	350	382	421	381	23.0	9.9	16.8

$n = 300$ $e = 1500$ % difference

Problem number	1	2	3	4	5	High	Low	Med.
Simplex	1697	1861	1770	1698	1736			
Penalty, $\alpha = 200$	1318	1362	1304	1228	1315	27.7	22.3	26.3
Penalty, $\alpha = 250$	1305	1409	1279	1264	1292	27.7	23.1	25.6

$n = 500$ $e = 2500$ % difference

Problem number	1	2	3	4	5	High	Low	Med.
Simplex	3453	3174	3434	3344	3230			
Penalty, $\alpha = 200$	2427	2294	2299	2376	2225	33.1	27.7	29.7
Penalty, $\alpha = 250$	2341	2291	2374	2287	2297	32.2	27.8	30.9

$n = 700$ $e = 3500$ % difference

Problem number	1	2	3	High	Low	Med.
Simplex	5222	5132	5177			
Penalty, $\alpha = 200$	3434	3483	3561	34.2	31.2	32.1
Penalty, $\alpha = 250$	3416	3558	3440	34.6	30.7	33.5

Table 3

Acknowledgement

The authors would like to thank Dr. R. Bartels and Dr. C. Haff of the University of Waterloo for their helpful discussions during the course of this research.

References

[1] R.H. Bartels, A Penalty Linear Programming Method Using Reduced–Gradient Basis–Exchange Techniques, *Lin. Alg. and its Appl.* 29 (1980) 17–32

[2] A.R. Conn, Constrained Optimization Using a Nondifferentiable Penalty Function, *SIAM J. Num. Anal.* 10 (1973) 760–784

[3] A.R. Conn, Linear Programming Via a Nondifferentiable Penalty Function, *SIAM J. Num Anal.* 13 (1976) 145–154

[4] W.H. Cunningham, A Network Simplex Method, *Mathematical Programming* 11 (1976) 105–116

[5] W.H. Cunningham, Theoretical Properties of the Network Simplex Method, *Math. of O.R.* 4 (1979) 196–208

[6] W.H. Cunningham and J.G. Klincewicz, On Cycling in the Network Simplex Method, *Mathematical Programming* 26 (1983) 182–189

[7] J. Edmonds and R.M. Karp, Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems, *Journal of the ACM* 19 (1972) 248–264

[8] L.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962

[9] D.R. Fulkerson, An Out–Of–Kilter Method for Minimum–Cost Flow Problems, *SIAM J. on Applied Mathematics* 9 (1961) 18–27

[10] F. Glover, D. Karney and D. Klingman, Implimentation and Computational Comparisons of Primal, Dual, and Primal–Dual Computer Codes for Minimum Cost Network Flow Problems, *Networks* 4 (1974) 191–212

[11] M.D. Grigoriadis, An Efficient Implimentation of the Network Simplex Method, *Math. Prog. Stud.* 26 (1986) 83–111

[12] J.L. Kennington and R.V. Helgason, *Algorithms for Network Programming*, John Wiley and Sons, 1980

[13] J.B. Orlin, Genuinely Polynomial Simplex and Non–Simplex Algorithms for the Minimum Cost Flow Problems, *Tech. Report, Sloan W.P. No.* 1615–84, (1984), M.I.T.

[14] E. Tardos, A Strongly Polynomial Minimum Cost Circulation Algorithm, *Combinatorica* 5 (1985) 247–256