A Network Simplex Algorithm for the Equal Flow Problem on a Generalized Network

David R. Morrison, Jason J. Sauppe, Sheldon H. Jacobson
Department of Computer Science University of Illinois, Urbana-Champaign, 201 N. Goodwin Ave., Urbana, Illinois 61801, USA, {morris50@illinois.edu, sauppe1@illinois.edu, shj@illinois.edu}

A network simplex algorithm is described for the minimum-cost network flow problem on a generalized network with the additional constraint that there exist sets of arcs which must carry equal amounts of flow. This problem can be modeled as a linear programming problem and solved using the standard simplex algorithm. However, due to the structure of the problem, more efficient algorithms are possible that solve the problem by operating directly on the network itself. One such algorithm is described that leads to improved asymptotic performance per iteration over the standard simplex algorithm, as long as the number of side constraints is small relative to the size of the network. Computational results are given comparing this algorithm to CPLEX's primal simplex solver on randomly-generated graphs.

Key words: generalized network flows; equal flow sets; side constraints; optimization; linear programming;
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received May 2010; revised January 2011, May 2011, July 2011; accepted July 2011.

---

1. Introduction and Background

This paper introduces a network simplex algorithm for computing the minimum-cost flow on a generalized network, with the additional constraint that there are sets of arcs for which the flow over these arcs must be equal. The problem is a combination of two well-studied network problems: the equal flow problem on pure networks (Calvete, 2003), and the generalized network flow problem (Ahuja et al., 1993).

The equal flow problem on pure networks was first considered by Ali et al. (1988), where they imposed the restriction that each of the equal flow sets have at most two arcs. Ahuja et al. (1999) consider the *simple* equal flow problem, in which there is exactly one set of equal flow arcs of arbitrary size; this problem is generalized by Calvete (2003), who proposes a network simplex algorithm for the *general* equal flow problem in which there can be an

arbitrary number of equal flow sets of arbitrary size. To avoid confusion, this problem is referred to as the *equal flow problem*. The equal flow problem can be solved by an adapted simplex algorithm that is more efficient per iteration than maintaining a simplex tableau, as long as the number of equal flow sets is not too large relative to the size of the network.

The generalized network problem is described in Ahuja et al. (1993, chap. 15). A generalized network has associated multiplicative factors for each arc in the network that cause the arc to either leak or produce extra flow in proportion to the amount sent from the arc's source. The minimum-cost flow problem can be defined analogously for generalized networks, and solved by an efficient combinatorial algorithm (Ahuja et al., 1993, chap. 15). A modification of the generalized network problem called the manufacturing network flow problem is described by Venkateshan et al. (2008), in which various types of resources must be made in similar proportions to each other.

Generalized networks appear in a wide range of practical applications. The aforementioned manufacturing process is one such application. Others include water distribution and management over leaky pipes, currency exchange networks (where the multipliers on the arcs represent the amount of money gained or lost when trading from one currency to another), or an energy network or chemical process, where the multipliers represent the efficiency of converting between two types of energy.

Meyers and Schulz (2009) provide a number of applications for the equal flow problem. In particular, they consider an integer version of the equal flow problem, in which the flow on every arc must be integral. They show this problem to be NP-hard and give a number of practical applications, including airline crew scheduling, supply chain management, and decoding of particular types of codes.

This paper discusses a hybrid algorithm between the algorithms for the generalized network flow problem and the equal flow problem. In Section 2, the equal flow problem on a generalized network is formally defined, and further terminology is established. Section 3 describes the basis structure needed for the network simplex algorithm, and shows that it is unique. Section 4 gives a procedure for computing flow on a network, given a particular basis structure. The optimality conditions for the algorithm are given in Section 5, and a generic pivoting rule for the algorithm is described in Section 6. Section 7 discusses the complexity of the algorithm. In Section 8, some computational results are presented, together with a comparison to CPLEX's primal simplex solver. Finally, Section 9 offers some concluding remarks and future directions. An appendix is included that contains proofs of all theorems

and lemmas.

2. Definitions and Terminology

The *equal flow problem on a generalized network* is defined on a directed graph $G = (N, A)$ where N is the set of nodes and A is the set of (directed) arcs in the graph. A supply vector b is given with entries for each node $i \in N$, where b_i represents the amount of supply available at node i. If $b_i > 0$, node i is a source; if $b_i < 0$, node i is a sink, and if $b_i = 0$, node i is a transport node (note that since G is a generalized network, the elements of the supply vector will not, in general, sum to 0).

Each arc $(i, j) \in A$, has an associated cost c_{ij}, an associated capacity $u_{ij} > 0$, and an associated multiplicative factor $\mu_{ij} \geq 0$. The multiplicative factor is such that if x_{ij} units of flow are sent from node i to node j along arc (i, j), then $\mu_{ij}x_{ij}$ units of flow arrive at node j. If $\mu_{ij} > 1$, then arc (i, j) is **gainy**, and if $\mu_{ij} < 1$, then arc (i, j) is **lossy**. Otherwise, $\mu_{ij} = 1$ and arc (i, j) is **breakeven**.

Furthermore, a collection $\mathcal{P} = \{R_1, R_2, \ldots, R_p\}$, $R_r \subseteq A$, $r = 1, 2, \ldots, p$ of **equal flow sets** is given. The goal is to find a feasible flow function $f : A \to \mathbb{R}$ of minimum cost that satisfies the demand at every vertex, and satisfies the additional property that for each $R_r \in \mathcal{P}$, $f_{ij} = f_{k\ell}$ for all $(i, j), (k, \ell) \in R_r$.

The equal flow problem on a generalized network can be modeled as a linear program with the introduction of some additional notation. For $i \in N$, define

$$d_r(i) = \sum_{(i,j)\in R_r} 1 - \sum_{(j,i)\in R_r} \mu_{ji}$$

as the net change in flow at node i if 1 unit of flow travels along the arcs in R_r.

Additionally, for each $R_r \in \mathcal{P}$, define $c_r = \sum_{(i,j)\in R_r} c_{ij}$, and $u_r = \min_{(i,j)\in R_r}\{u_{ij}\}$. Lastly, define $A' = \{(i, j) \in A \mid (i, j) \notin \cup_{r=1}^{p} R_r\}$. Then the linear programming formulation of the equal flow problem on a generalized network is:

$$\text{minimize} \quad \sum_{(i,j)\in A'} c_{ij}x_{ij} + \sum_{r=1}^{p} c_r x_r \tag{1a}$$

$$\text{subject to} \quad \sum_{j:(i,j)\in A'} x_{ij} - \sum_{j:(j,i)\in A'} \mu_{ji}x_{ji} + \sum_{r=1}^{p} d_r(i)x_r = b_i \qquad \text{for all } i \in N \tag{1b}$$

$$0 \le x_{ij} \le u_{ij} \qquad \text{for all } (i,j) \in A' \tag{1c}$$

$$0 \le x_r \le u_r \qquad \text{for all } r = 1, 2, ..., p \tag{1d}$$

The above linear program has an embedded network structure given by constraints (1b) and (1c), together with some non-network variables and side constraints given by line (1d). Because of the embedded network structure in the LP, specialized algorithms can be developed that operate directly on the network and require less computational overhead than the standard simplex algorithm. This paper introduces one such algorithm, GenNetEq, which has an improved worst-case running time than the standard simplex algorithm in most cases, provided that p is small relative to the number of nodes in the graph (if p is not small relative to $|N|$, the basis structure used by GenNetEq may consist entirely of equal flow sets, eliminating the ability to exploit the network structure).

3. Basis Structure

Assuming that an instance of the equal flow problem on a generalized network has a feasible solution, the constraint matrix associated with (1) has rank $n = |N|$. Thus, a basis for the problem instance has n basic variables with the remaining variables at either their upper or lower bounds. GenNetEq maintains a basis structure $(\mathbf{B}, \mathbf{L}, \mathbf{U})$, where the sets $\mathbf{B}$, $\mathbf{L}$, and $\mathbf{U}$ contain the arcs and equal flow sets whose corresponding variables are basic, non-basic at their lower bound, or non-basic at their upper bound, respectively. To simplify notation, the members of the sets $\mathbf{B}, \mathbf{L}$, and $\mathbf{U}$ will be used interchangeably as either flow variables or the corresponding arcs and equal flow sets of the network G. The basic arcs of $\mathbf{B}$ are $\mathbf{B} \cap A'$; the remaining members of $\mathbf{B}$ are basic equal flow sets.

The members of $\mathbf{B}$ form a basis for (1) allowing the flow on G to be uniquely determined. This implies that the corresponding columns in the constraint matrix are linearly independent. Furthermore, any maximal set of linearly independent columns in the constraint matrix induces a particular structure on the graph itself, so that any basis structure $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ which maintains this graph structure will have a unique associated flow solution.

This solution can be determined for any basis structure $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ by setting all variables in $\mathbf{L}$ and $\mathbf{U}$ to their lower and upper bounds respectively, and then computing the flows for the variables in $\mathbf{B}$. The specific structure of $\mathbf{B}$ is described next, and a proof is given that demonstrates that such a structure enables flow to be uniquely determined.

3.1. $\mathcal{S}$-Augmented Spanning Forests

The set $\mathbf{B}$ maintained in the basis structure used by GenNetEq is a modification of the augmented spanning forest used in the generalized network flow problem. For that problem, the basis structure consists of a collection of *augmented trees* that span the entire network:

Definition 1 (Ahuja et al. 1993)**.** *An* **augmented tree** *is a tree T together with an extra arc (α, β), where $\alpha, \beta \in T$. The cycle formed by the arc (α, β) is the* **extra cycle***, and is denoted W. Define the quantity*

$$\mu(W) = \frac{\prod_{(i,j)\in\overline{W}} \mu_{ij}}{\prod_{(i,j)\in\underline{W}} \mu_{ij}}$$

where $\overline{W}$ is the set of arcs oriented in the same direction as W, and $\underline{W}$ is the set of arcs oriented in the opposite direction of W. For an augmented tree $T + (\alpha, \beta)$, if $\mu(W) \neq 1$, the augmented tree is **good***.*

The basic arcs in GenNetEq form an augmented spanning forest in the underlying network, and the basic equal flow sets form connections between the components (not necessarily all) in the forest. This structure is referred to as an **$\mathcal{S}$-augmented spanning forest**, where $\mathcal{S} \subseteq \mathcal{P}$ is a collection of basic equal flow sets. For notational convenience, assume that $R_1, R_2, ..., R_s$ are the basic equal flow sets while $R_{s+1}, ..., R_p$ are the non-basic equal flow sets. This spanning forest consists of good augmented trees as well as $|\mathcal{S}|$ other (non-augmented) trees. Intuitively, the extra arc for these non-augmented trees is an equal flow set, though it is not necessary to actually maintain an association of equal flow sets with any particular tree. The good augmented trees in the spanning forest are referred to as type I trees. The s remaining trees will be referred to as type II trees and will be denoted as $\overline{T_1}, \overline{T_2}, ..., \overline{T_s}$.

The basis structure is a hybridization of the basis structure used by Calvete (2003) for the equal flow problem and the augmented spanning forest basis structure used in the generalized network flow problem (Ahuja et al., 1993), and is given in Definition 2.

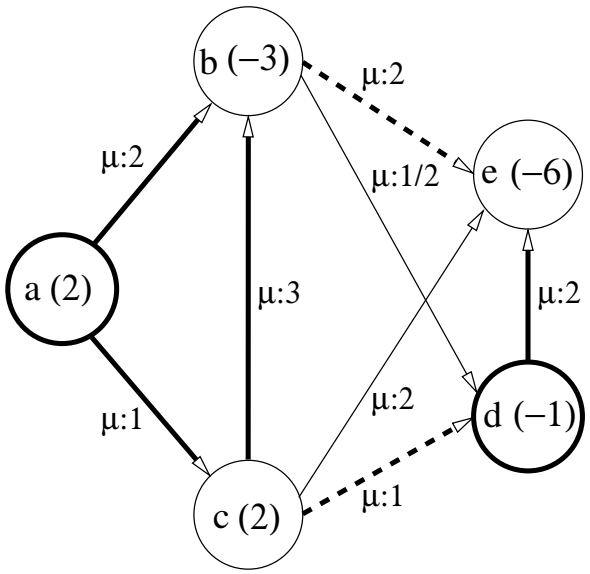

Figure 1: A good $\mathcal{S}$-augmenting forest for this network is shown by the bold arcs. The supply at each node is specified in parentheses, and the gain/loss factor μ is listed next to each arc. The dashed arcs belong to an equal flow set. Costs and upper bounds are not shown. The flow corresponding to this basis is 2 on the equal flow arcs and 1 on all other basic arcs. The non-basic arcs have 0 flow.

Definition 2. *A* **good $\mathcal{S}$-augmented forest** *is a set of $|\mathcal{S}|$ type II trees $\overline{T_1}, \overline{T_2} \ldots, \overline{T_{|\mathcal{S}|}}$, together with k type I trees $T_1 + (\alpha_1, \beta_1), T_2 + (\alpha_2, \beta_2), \ldots, T_k + (\alpha_k, \beta_k)$, that collectively span G (k may be 0). Furthermore, the submatrix formed by the rows and columns of the constraint matrix associated with the type II trees, together with the columns corresponding to equal flow sets, must have full rank. The set N_I is the set of nodes spanned by the type I trees, and the set N_{II} is the set of nodes spanned by type II trees.*

An example network is presented in Figure 1, together with a good $\mathcal{S}$-augmented forest. In this example, the basis structure consists of one type I tree and one type II tree, together with a single equal flow set. The type I tree is rooted at node a, and has extra arc (c, b). The type II tree is rooted at node d; it is trivial to verify that the submatrix of rows and columns corresponding to the equal flow set $\{(b, e), (c, d)\}$ has full rank, so the basis structure shown is valid.

3.2. Finding an Initial Basis

As will be shown, if a feasible solution to the equal flow problem on a generalized network exists, then a corresponding good $\mathcal{S}$-augmented forest exists. To find such a structure on an

arbitrary problem instance may be a non-trivial task, however. To initialize GenNetEq, several different standard network transformations can be used to find a good $\mathcal{S}$-augmented forest. GenNetEq creates self loops at every node, similar to the approach of Ahuja et al. (1993) (See Figure 2).

The next section shows that any basis for the equal flow problem on a generalized network must have this structure. Further, a good $\mathcal{S}$-augmented forest allows flow values to be uniquely determined for all basic variables.

4. Flow Computation

Given a good $\mathcal{S}$-augmented forest structure $(\mathbf{B}, \mathbf{L}, \mathbf{U})$, the flow can be uniquely determined as follows. Initialize a modified supply vector $\hat{b} = b$. For each arc $(i, j) \in \mathbf{L}$, set the flow x_{ij} to 0. For each equal flow set $R_r \in \mathbf{L}$, set $x_r = 0$. For each arc $(i, j) \in \mathbf{U}$, set $x_{ij} = u_{ij}$, and for each equal flow set $R_r \in \mathbf{U}$, set $x_r = u_r$. Finally, for each arc $(i, j) \in \mathbf{U}$, update $\hat{b}$ by subtracting u_{ij} from $\hat{b}_i$ and adding $\mu_{ij} u_{ij}$ to $\hat{b}_j$, and for each equal flow set $R_r \in \mathbf{U}$, subtract $u_r d_r(i)$ from $\hat{b}_i$ for all $i \in N$.

Then, for each equal flow set $R_r \in \mathbf{B}$, set $x_r = \theta_r$. Traverse each type II tree from leaves to root to determine the flows on each arc in terms of the θ_r's. Next, solve the system of equations induced by the flow balance equations at the root of each type II tree. This system of equations must have a unique solution if the basis structure is valid, since this system is precisely encoded by the constraint matrix of the type II trees and basic equal flow sets. Once this solution has been determined, the remaining flow values on the arcs in the type II trees can be determined by substituting in the θ_r values. After this is done, the supply at all remaining nodes $i \in N_I$ is updated by adding $d_r(i) x_r$ to $\hat{b}_i$ for each $R_r \in \mathbf{B}$.

Finally, for every type I tree $T_i + (\alpha_i, \beta_i)$ in the spanning forest, set the flow on (α_i, β_i) equal to θ. By traversing the tree from leaves to root, the flow on each arc can be determined in terms of θ. The flow balance constraints at the root imply a unique solution for θ (this is precisely the calculation performed to obtain flows in the generalized network setting; see (Ahuja et al., 1993) for more details).

This procedure is summarized in Algorithm 2, ComputeFlows. At the end of this procedure, flow values have been assigned for all arcs in A' and for all equal flow sets. This process is deterministic, so it uniquely determines these flow values. Furthermore, Theorem 1 establishes that this is the only possible basis structure.

Theorem 1. *The basis structure* $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ *has a corresponding unique flow solution where* $\mathbf{B}$ *forms a good* $\mathcal{S}$*-augmented forest for* $\mathcal{S} = \{R_r : R_r \in \mathbf{B}\}$*. Furthermore, any basis for (1) must form a good* $\mathcal{S}$*-augmented forest.*

5. Optimality Conditions

The basic step for pivoting in GenNetEq is no different than in the pure network simplex algorithm: calculate the reduced costs of all non-basic arcs and equal flow sets, choose a sub-optimal arc or equal flow set to enter the basis, and then determine how much flow can be pushed across that arc or equal flow set until some basic variable becomes saturated (reaches its upper or lower bound) and leaves the basis (note that the entering and leaving variable could be the same).

In order to calculate the reduced costs, node potentials must be determined. Node potentials are precisely the dual variables of (1b), and are denoted by π_i for $i \in N$. The reduced cost (with respect to a set of node potentials $\pi = \{\pi_i\}_{i \in N}$) of an arc $(i, j) \in A'$ is defined as:

$$c_{ij}^{\pi} = c_{ij} - \pi_i + \mu_{ij}\pi_j.$$

The reduced cost for an equal flow set R_r is defined as:

$$\begin{aligned} c_r^{\pi} &= c_r - \sum_{(i,j) \in R_r} (\pi_i - \mu_{ij}\pi_j) \\ &= c_r - \sum_{i \in N} d_r(i)\pi_i \end{aligned}$$

Note that c_r^{π} is simply the sum of the reduced costs of all arcs in R_r, and is precisely the reduced cost of x_r in the linear programming model (1).

Minimizing $c^T x$ is equivalent to minimizing $(c^{\pi})^T x$, where $c^T x = \sum_{(i,j) \in A'} c_{ij}x_{ij} + \sum_{r=1}^{p} c_r x_r$ (similarly for $(c^{\pi})^T x$) (Ahuja et al., 1993). Theorem 2 gives the optimality conditions for a flow solution x^*:

Theorem 2. *If a flow* x^* *is a feasible solution for the equal flow problem on a generalized network and* x^* *satisfies the following conditions for some set of node potentials* π*, then it is*

optimal:

$$\begin{array}{ll} (1a) & \textit{If } 0 < x^*_{ij} < u_{ij}, \textit{ then} \quad c^\pi_{ij} = 0. \\ (1b) & \textit{If } 0 < x^*_r < u_r, \textit{ then} \quad c^\pi_r = 0. \\ (2a) & \textit{If } x^*_{ij} = 0, \textit{ then} \quad c^\pi_{ij} \geq 0. \\ (2b) & \textit{If } x^*_r = 0, \textit{ then} \quad c^\pi_r \geq 0. \\ (3a) & \textit{If } x^*_{ij} = u_{ij}, \textit{ then} \quad c^\pi_{ij} \leq 0. \\ (3b) & \textit{If } x^*_r = u_r, \textit{ then} \quad c^\pi_r \leq 0. \end{array} \tag{2}$$

This result leads to the following optimality conditions for a good $\mathcal{S}$-augmented forest structure:

Corollary 1. *A good $\mathcal{S}$-augmented forest structure $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ with an associated feasible flow solution x^* is optimal if for some set of node potentials π, the optimality conditions in (2) are satisfied.*

These optimality conditions lead to a method for computing a set of node potentials π for a given basis structure $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ with feasible flow x: construct π so that $c^\pi_{ij} = 0$ for all basic (i, j) and $c^\pi_r = 0$ for all basic R_r, then determine values for the remaining π's. If none of the above optimality constraints are violated for the remaining arcs, then $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ is optimal. Otherwise, a violated optimality condition has been identified and a pivoting operation can be performed.

To compute the node potentials, set the value of $\pi_h = \theta$ for each type I tree $T + (\alpha, \beta)$ in the $\mathcal{S}$-augmented forest, where h is the root of T. Then, solve $0 = c_{ij} - \pi_i + \mu_{ij}\pi_j$ for each arc in T (since T has a tree structure, this can be done uniquely in terms of θ). Lastly, solve $0 = c_{\alpha\beta} - \pi_\alpha + \mu_{\alpha\beta}\pi_\beta$ to determine a numerical value for θ (note that this procedure is identical to the procedure for the generalized network setting).

Next, for each type II tree $\overline{T_i}$ in the $\mathcal{S}$-augmented forest, set the value of $\pi_{h_i} = \theta_i$, where h_i is $\overline{T_i}$'s root (chosen arbitrarily). As before, solve for all arcs in $\overline{T_i}$ in terms of θ_i. Next, solve the following system of s equations to determine numerical values for each θ_i:

$$0 = c_r - \sum_{v \in N} d_r(v)\pi_v \qquad \text{for all } r = 1, 2, \ldots, s \tag{3}$$

This system of equations will have a unique solution, since the submatrix formed by the rows and columns of the constraint matrix associated with the type II trees has full rank, by definition. Thus by using this technique, unique node potentials can be determined for all nodes in the type II trees. This procedure is summarized in Algorithm 3, ComputePotentials.

6. Pivoting

During each iteration, GenNetEq moves from one feasible basis to another by pivoting. In this process, a variable is selected to enter the basis, and a leaving variable is identified and removed from the basis.

6.1. Selecting an Entering Variable

Once all node potentials have been determined, reduced costs for all non-basic arcs and equal flow sets can be computed according to the above optimality conditions. If any condition is violated, then the variable corresponding to the offending arc or equal flow set can be chosen to enter the basis. In the case of multiple violations, the entering variable can be chosen using any pivoting scheme (largest violation, lowest index, etc.), though different choices may lead to different rates of convergence. Experimental results suggest that a largest-violation rule where the reduced cost of the equal flow sets is scaled by the number of arcs in the set produces fast convergence in terms of number of iterations, and was not too computationally expensive. Alternatively, a candidate list approach as described in Ahuja et al. (1993, chap. 11) gives substantial savings in CPU time without sacrificing too many iterations. Once an entering variable has been determined, a leaving variable must be determined so that the pivoting operation can be performed.

6.2. Selecting a Leaving Variable

In this section, the entering variable will be assumed to be at its lower bound; a similar procedure can be applied when the entering variable is at its upper bound. In order to determine the total allowable change δ in the entering variable's flow, one unit of flow can be pushed on this variable to determine the relative (linear) rates of change $\Delta_{x_\mathbf{B}}$ in the flows on the basic variables. These rates of change can be used to determine which basic variable will be saturated first as the flow on the entering variable is increased. It is also possible that the flow on the entering variable can be increased to its upper bound, in which case the entering variable is also the leaving variable.

The entering variable may either correspond to an arc (k, l) or to an equal flow set R_q. There are several cases to consider:

Case 1: $k \in T_e, l \in T_f$ **for some type I trees** $T_e + (\alpha, \beta)$**,** $T_f + (\gamma, \eta)$

There are two sub-cases: Either $T_e = T_f$, or they are different trees. In either case, the pivoting procedure for the generalized network simplex algorithm described by Ahuja et al. (1993) can be performed exactly; no modification is necessary, so this will not be described here.

Case 2: $k \in \overline{T_e}, l \in \overline{T_f}$ for some type II trees $\overline{T_e}$, $\overline{T_f}$

When the entering arc forms a connection between two type II trees, sending flow along that arc will decrease the total supply available in $\overline{T_e}$ and increase the total supply available in $\overline{T_f}$. The maximum amount of flow that can be sent along arc (k, l) can be determined as follows:

1. Send one unit of flow across the arc (k, l).
2. Run procedure ComputeFlows to determine the relative rates of change $\Delta_{x_\mathbf{B}}$ on all basic variables, with respect to the new supply created by pushing flow across (k, l).
3. Find the variable that will reach its upper or lower bound first (called the blocking variable) as flow is pushed along (k, l).

This procedure will return a value δ that can be pushed along arc (k, l) that will maintain feasibility. It also produces a leaving arc (i, j) or a leaving basic equal flow set R_h that will leave the basis at its upper or lower bound.

Case 3: $k \in T_e, l \in \overline{T_f}$ or $k \in \overline{T_f}, l \in T_e$ for a type I tree $T_e + (\alpha, \beta)$ and a type II tree $\overline{T_f}$

In this case, pushing flow along arc (k, l) increases or decreases the available supply in $T_e + (\alpha, \beta)$ while doing the opposite to the supply in $\overline{T_f}$. Intuitively, $T_e + (\alpha, \beta)$ can generate or consume supply as needed by pushing it around its extra cycle, but $\overline{T_f}$ needs to disperse or obtain the additional supply from elsewhere in the graph via the equal flow sets. As in Case 2, a single unit of flow is sent across (k, l), and ComputeFlows solves for the relative rates of change on all basic variables. This allows a blocking variable (i, j) or R_h to be identified and leave the basis.

Case 4: The entering variable corresponds to an equal flow set R_q

In this case, an additional equal flow set is selected to distribute flow between the trees in the forest. Since the size of the basis must remain unchanged, bringing this equal flow set into the basis can either replace one of the current basic equal flow sets, replace a basic arc

in a type II tree (thus splitting the type II tree into two disjoint type II trees), or replace a basic arc in a type I tree (thus either splitting the type I tree into a smaller type I tree and a type II tree or transforming the type I tree into a type II tree by breaking the extra cycle). To determine which possibility occurs, it is necessary to compute relative rates of change for the flows on the basic variables per unit change in the flow on R_q and then identify a variable to leave the basis.

The pivoting process proceeds in a similar fashion to Cases 2 and 3. By pushing a single unit of flow across all arcs in the equal flow set, the ComputeFlows procedure can determine a unique flow solution $\Delta_{x_\mathbf{B}}$. From this, as in the other cases, a blocking variable (i, j) or R_h can be identified to leave the basis.

This pivoting process moves from one feasible basis of (1) to another. Since any basis for (1) forms an $\mathcal{S}$-augmented forest, pivoting moves between $\mathcal{S}$-augmented forest structures with feasible flow. After a pivoting operation, new flow values have been computed for all basic variables, and node potentials will need to be updated using ComputePotentials. Pseudocode for the pivoting procedure is given in Algorithm 4, Pivot. For an example of the pivoting process, see Figure 3.

7. Complexity of GenNetEq

The complexity of the GenNetEq algorithm depends on two factors: the complexity per iteration and the total number of iterations needed before the algorithm terminates.

7.1. Per-Iteration Complexity

The complexity of an iteration can be determined by observing that each iteration requires a call to ComputePotentials, the selection of an entering variable or a determination that no variable violates its optimality conditions, and then a pivoting operation which updates the basis and flow solution x. Let $n = |N|$, $m' = |A'|$, and $p = |\mathcal{P}|$.

First, note that the ComputeFlows procedure is an efficient algorithm to determine the flow for a basis structure $(\mathbf{B}, \mathbf{L}, \mathbf{U})$. This procedure fixes variables in $\mathbf{L}$ and $\mathbf{U}$ to their bounds, then traverses every arc in $\mathbf{B}$ at most twice, and finally sets up and solves a system of at most p equations. It takes $O(np)$ time to set up this system, and $O(p^3)$ time to solve it with Gaussian elimination, so ComputeFlows runs in $O(m' + np + p^3)$ time. Similarly,

ComputePotentials must set up and solve a system of equations of size at most p, and it visits every node in the graph at most twice, so it takes $O(np + p^3)$ time.

Once node potentials are determined, Pivot is called. This procedure has three steps: selecting an entering variable (if one exists), finding a leaving variable, and updating the basis structure. To find an entering variable, the reduced costs of all non-basic variables are computed, and one is selected that violates its optimality conditions. This can be done in $O(m' + np)$ time. Finding a leaving variable can be done in at most $O(np + p^3)$ time, and the remaining work to update the basis structure takes $O(n)$ time.

Therefore, the per-iteration running time of GenNetEq is $O(m' + np + p^3)$. This should be contrasted with the worst-case running time of the simplex algorithm: both the full tableau implementation and the revised simplex implementation require $O(nm' + np)$ time per iteration for a linear program with n rows and $m' + p$ variables (Bertsimas and Tsitsiklis, 1997).

When $p = O(\sqrt{n})$, the per-iteration running time of GenNetEq simplifies to $O(m + n^{3/2})$, which becomes linear in m when $m = \Omega(n^{3/2})$. This is a clear improvement over the worst-case per-iteration running time of the standard simplex algorithm. Furthermore, the per-iteration asymptotic running time of GenNetEq is identical to the algorithm described by Calvete (2003). GenNetEq's per-iteration running time is slightly slower than the $O(m)$ per-iteration running time of the simplex algorithm for generalized networks.

7.2. Termination

When an iteration of GenNetEq results in a pivot with $\delta > 0$, then the cost of the flow will decrease after that iteration. Assuming that all arcs have finite capacity, the value of the optimal cost is bounded below. Thus, GenNetEq will terminate after a finite number of iterations, assuming each iteration is non-degenerate (i.e., produces a non-zero improvement). Unfortunately, empirical evidence suggests that upwards of 90% of pivots in a network LP are degenerate (Elam et al., 1979). However, since GenNetEq operates in the same manner as the standard simplex algorithm, finite termination can be achieved by applying an appropriate perturbation to the initial supply vector b. Moreover, perturbation techniques can also be used to improve the overall performance of the algorithm by promoting sparse pivots (see, for example, Bixby 2002).

Even though GenNetEq will terminate using perturbations, the total number of pivots could still be exponential (just as in the simplex algorithm). However, it is suspected

that most problems will require a number of pivots that is polynomial in m and n. This performance was observed for the experiments that were run.

8. Computational Results

An implementation of GenNetEq was written in C++, and run on a variety of problem instances to gain additional insight into the problem. The problem instances were generated randomly, using a modification of the GNETGEN algorithm developed by Klingman et al. (1974) and Clark et al. (1992). The network generator behaves identically to GNETGEN, except that it also allows for placement of equal flow sets to maintain network feasibility. Some example instances, together with the code and parameter settings used to generate all graphs, is provided in the online supplement.

Each problem instance was solved by GenNetEq on a single core of an Intel Core i7-930 2.8GHz quad-core processor, with 12 GB of available memory (a maximum of 1 GB was actually used in the experiments). Trials were run on 1200-node networks, across networks ranging from 35 970 to 647 460 total arcs. Trials were done with both 10 and 50 equal flow sets, with 30% of the total number of arcs distributed equally among all equal flow sets. 30 experiments were performed for all combinations of parameters.

Results were then compared to the primal simplex solver and the network simplex solver in CPLEX. Experiments show that CPLEX's network solver is unable to extract an appropriate network structure for the generalized equal flow problem, and so resorts to either the primal or dual LP simplex solver. Thus, the results from the network simplex solver are not shown here. CPLEX's primal simplex solver was invoked with preprocessing disabled and the candidate list pricing scheme (partial pricing) turned on, as this provides the fairest comparison between CPLEX and GenNetEq.

Table 1 shows statistics for GenNetEq when run across all experiments. GenNetEq takes just under 1.5 minutes to solve the largest problem instances (647 460 arcs and 50 equal flow sets). Table 2 reports the results of the comparison between GenNetEq and CPLEX as the geometric mean of ratios of GenNetEq and CPLEX on all problem instances, grouped by problem size and number of equal flow sets. GenNetEq takes about three-quarters as many iterations as CPLEX does on all problem instances, but with respect to CPU seconds, GenNetEq takes several times as long.

The performance of CPLEX is unsurprising on this problem, given that CPLEX is a

10 equal flow sets				50 equal flow sets			
Arcs	**Iterations**	**CPU(s)**	**sec/iter**	**Arcs**	**Iterations**	**CPU(s)**	**sec/iter**
35 970	7500	2.3	0.00030	35 970	10 000	5.2	0.00049
71 940	10 000	3.9	0.00038	71 940	14 000	7.4	0.00054
143 880	16 000	8.0	0.00050	143 880	19 000	13.	0.00067
215 820	21 000	12.	0.00058	215 820	25 000	20.	0.00078
287 760	25 000	16.	0.00062	287 760	32 000	27.	0.00085
359 700	32 000	21.	0.00066	359 700	39 000	35.	0.00091
431 640	39 000	28.	0.00071	431 640	47 000	45.	0.00097
503 580	45 000	33.	0.00073	503 580	55 000	55.	0.0010
575 520	53 000	41.	0.00078	575 520	62 000	66.	0.0011
647 460	62 000	50.	0.00080	647 460	74 000	81.	0.0011

Table 1: Results showing the average absolute number of iterations, total running time, and time per iteration taken by GenNetEq with respect to the number of arcs and equal flow sets in the network.

10 equal flow sets				50 equal flow sets			
Arcs	**Iterations**	**CPU (s)**	**sec/iter**	**Arcs**	**Iterations**	**CPU (s)**	**sec/iter**
35 970	0.63	6.2	9.8	35 970	0.70	7.6	11.
71 940	0.60	6.2	10.	71 940	0.70	7.3	10.
143 880	0.59	6.4	11.	143 880	0.62	6.1	9.9
215 820	0.55	5.3	9.5	215 820	0.62	5.2	8.4
287 760	0.55	4.1	7.5	287 760	0.65	4.3	6.6
359 700	0.57	4.9	8.5	359 700	0.64	4.8	7.5
431 640	0.58	3.9	6.8	431 640	0.62	3.9	6.2
503 580	0.56	4.5	7.9	503 580	0.60	4.7	7.8
575 520	0.57	4.7	8.3	575 520	0.61	4.4	7.1
647 460	0.59	4.8	8.0	647 460	0.65	4.8	7.4

Table 2: Results showing the comparison of GenNetEq to CPLEX, with respect to the number of arcs and equal flow sets in the network. Results in each column are given as $\sqrt[30]{\prod_{i=1}^{30} \text{GenNetEq}(i) / \prod_{i=1}^{30} \text{CPLEX}(i)}$, where the products are taken over each instance i for a given set of parameters.

highly-optimized commercial software library. In Table 3, a profile analysis of the specific implementation of GenNetEq is presented, to demonstrate where improvements could be made in this implementation. Five instances from each class of problems were chosen to be profiled using the `gprof` Linux utility; the results for each major subroutine of GenNetEq were averaged together to provide insight as to where the bulk of the work is being performed.

As can be seen, the largest portion of running time is spent in selecting an entering variable, and performing all steps necessary to pivot out the leaving variable. Closer analysis of the profiling results shows that about two-thirds of the work in Pivot is being spent updating the basis, which suggests that further optimization of the tree-updating procedure may improve the running time.

Select	Pivot	ComputePotentials	ComputeFlows	Other
47.56%	42.17%	8.13%	1.64%	0.5%

Table 3: Results of profiling the code for an implementation of GenNetEq. The majority of the work is being performed in selection and pivoting. The Select procedure computes reduced costs and selects an entering variable.

Another interesting observation from these experiments is that in general, around 90% of the pivots performed by GenNetEq are between two type I trees. Thus, in most cases, GenNetEq does not need to perform any extra work above and beyond that done by the generalized network simplex algorithm. It may be possible to invoke a highly-optimized generalized network simplex solver for these pivots to gain additional improvements in running time.

9. Conclusion

The network simplex algorithm described in this paper solves the minimum-cost flow problem on generalized networks with equal flow sets. By exploiting the structure inherent in the network, the algorithm is able to gain improvements in theoretical per-iteration efficiency over the standard primal simplex algorithm, and has comparable worst-case per-iteration running times as other network simplex algorithms on related problems. This combinatorial structure is able to provide a number of important insights into the nature of this problem that are not immediately apparent from the linear programming formulation.

As such, this problem admits a number of directions for future research; first, other generalized network flow problems that have similar side constraints should be identified, and an adaptation of the basis structure of GenNetEq should be developed to solve these problems. This may be helpful to identify a general class of problems for which a network simplex-based approach is useful.

Finally, it should be seen if a dual network simplex algorithm for solving the equal flow problem on generalized networks can be developed. This algorithm would ignore capacity constraints and operate on variables with infeasible flow values. Work on dual algorithms for pure networks suggest that such an algorithm may exist, and using scaling techniques, may guarantee strongly polynomial running times for such an algorithm (see, for example, Armstrong and Jin 1997).

Appendix

A. Proof of Theorem 1

Proof. ($\Rightarrow$) This is established by the ComputeFlows procedure.

($\Leftarrow$) To show that any basis for (1) must form an $\mathcal{S}$-augmented spanning forest, suppose that there is some basis $\mathbf{B}'$ that does not. Then it must be the case that the flow function for this basis is not unique. Note that any basis must have the same size, which for this problem is $n = |N|$ (to see this, take $\mathcal{P} = \emptyset$, and the problem collapses to the generalized network setting). First, it will be shown that if the number of type II trees formed by the basic arcs in $\mathbf{B}'$ is not exactly $|\mathcal{S}|$, then either the size of $\mathbf{B}'$ is greater than n, or the flow function cannot be uniquely determined for $\mathbf{B}'$.

Let N_{II} be the subset of nodes spanned by the set of type II trees, with $b = |N_{II}|$, and let ℓ be the number of type II trees. Let X be the nodes in the remainder of the graph (i.e., $X = N \setminus N_{II}$, and $|X| = n - b$). This implies that there must be at least $n - b$ basic variables spanning the nodes in X, since there must be at least one cycle contained in every component of X. Thus, if $\ell < |S|$, then

$$|\mathbf{B}'| \geq n - b + b - \ell + |\mathcal{S}| > n - b + b - |S| + |S| = n,$$

which violates the condition that all bases have the same size.

Next, suppose that $\ell > |\mathcal{S}|$. Then, by a similar analysis as above, in order for $\mathbf{B}'$ to have enough basic variables, there must be at least one component of N_{II} which has more than one cycle. If all cycles of this component are breakeven, then flow around any cycle can be determined arbitrarily. On the other hand, if there exist two cycles W_1 and W_2 in this component which are not breakeven, we can orient W_1 so that it is gainy, and W_2 so that it is lossy. By the discussion of bicycles in Ahuja et al. (1993, pp. 587-589), an arbitrary amount of flow can be sent around W_1 and have the excess absorbed by W_2, so the flow on G is not unique.

Finally, suppose that $\ell = |\mathcal{S}|$ (i.e., the number of type II trees is exactly $|\mathcal{S}|$). Furthermore, everything in X must be of type I, or the flow on some component of X cannot be uniquely determined. That is, X is precisely N_I, and every tree in X must have a non-breakeven cycle, as shown by Ahuja et al. (1993, chap. 15). Thus, the submatrix corresponding to the type II trees of $\mathbf{B}'$, together with the basic equal flow sets, must not

have full rank. By definition, however, this means that the flow on G cannot be uniquely determined. ∎

B. Proof of Theorem 2

Proof. For x^* to be optimal, it must satisfy $c^T x^* \leq c^T x$ for all other feasible x. Equivalently, for some set of node potentials π that satisfy the conditions above, $(c^\pi)^T x^* \leq (c^\pi)^T x \rightarrow (c^\pi)^T (x - x^*) \geq 0$ for all feasible x. To show that x^* is optimal, then, it suffices to show that $(c^\pi)^T (x - x^*)$ is nonnegative. In fact, the stronger condition that each term of

$$(c^\pi)^T (x - x^*) = \sum_{(i,j) \in A'} c_{ij}^\pi \left(x_{ij} - x_{ij}^* \right) + \sum_{r=1}^{p} c_r^\pi \left(x_r - x_r^* \right)$$

is nonnegative will be shown. There are several cases to consider.

(1a) $0 < x_{ij}^* < u_{ij}$ implies that $c_{ij}^\pi = 0$, so it follows that $c_{ij}^\pi \left(x_{ij} - x_{ij}^* \right) = 0$.

(1b) $0 < x_r^* < u_r$ implies that $c_r^\pi = 0$, so it follows that $c_r^\pi \left(x_r - x_r^* \right) = 0$.

(2a) $x_{ij}^* = 0$ implies that $c_{ij}^\pi \geq 0$ and $x_{ij} \geq x_{ij}^*$, so it follows that $c_{ij}^\pi \left(x_{ij} - x_{ij}^* \right) \geq 0$.

(2b) $x_r^* = 0$ implies that $c_r^\pi \geq 0$ and $x_r \geq x_r^*$, so it follows that $c_r^\pi \left(x_r - x_r^* \right) \geq 0$.

(3a) $x_{ij}^* = u_{ij}$ implies that $c_{ij}^\pi \leq 0$ and $x_{ij} \leq x_{ij}^*$, so it follows that $c_{ij}^\pi \left(x_{ij} - x_{ij}^* \right) \geq 0$.

(3b) $x_r^* = u_r$ implies that $c_r^\pi \leq 0$ and $x_r \leq x_r^*$, so it follows that $c_r^\pi \left(x_r - x_r^* \right) \geq 0$.

This establishes that all terms are nonnegative, so it follows that $(c^\pi)^T x^* \leq (c^\pi)^T x$ for all feasible x. Therefore, x^* is an optimal solution to the equal flow problem on a generalized network. ∎

C. Figures

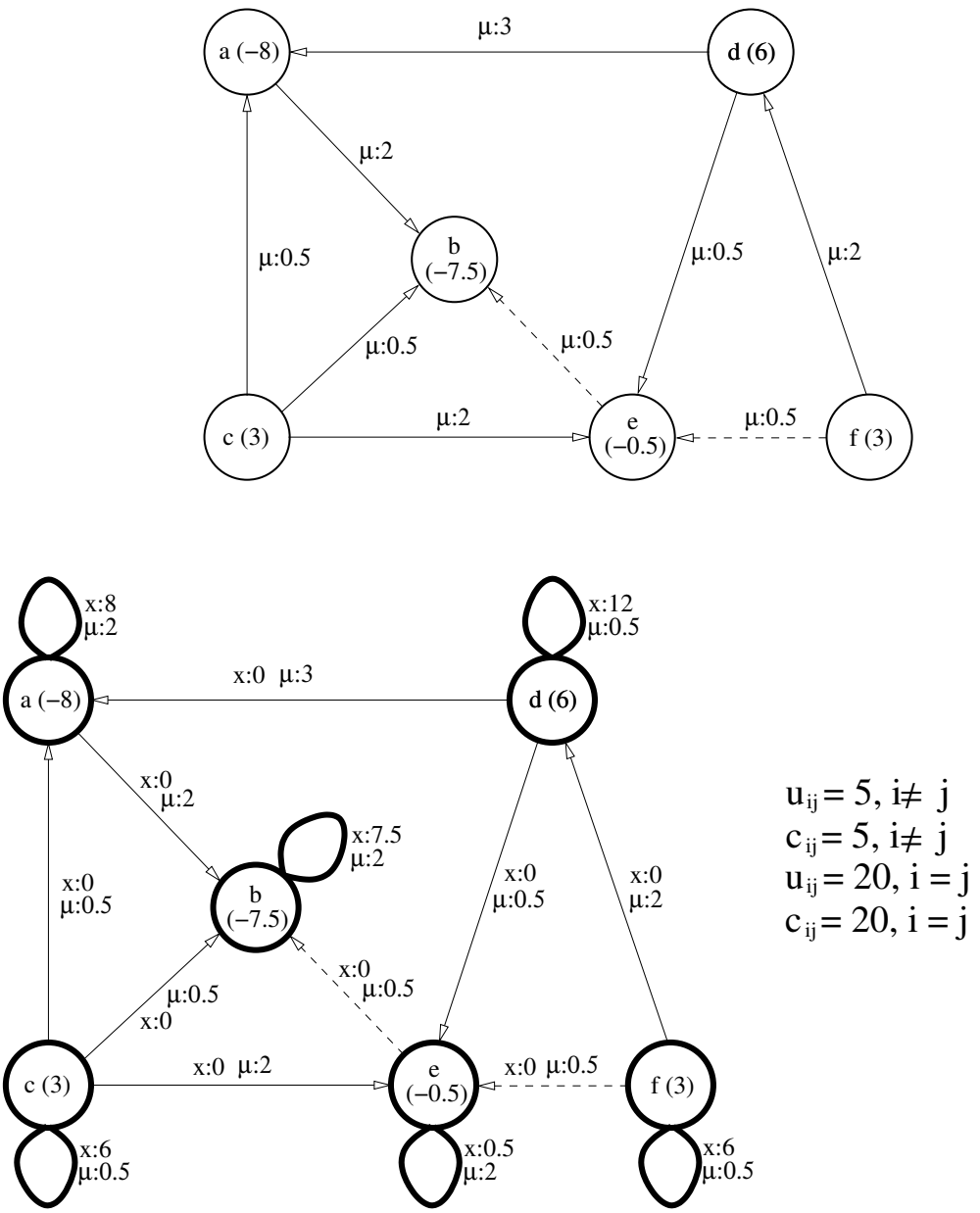

Figure 2: Forming an initial feasible solution for an example network. The network on the left is the original network, and the network on the right shows the result after adding self-loops to every node. The initial basis, shown in bold, consists of all type I trees with a single node and a self-loop.

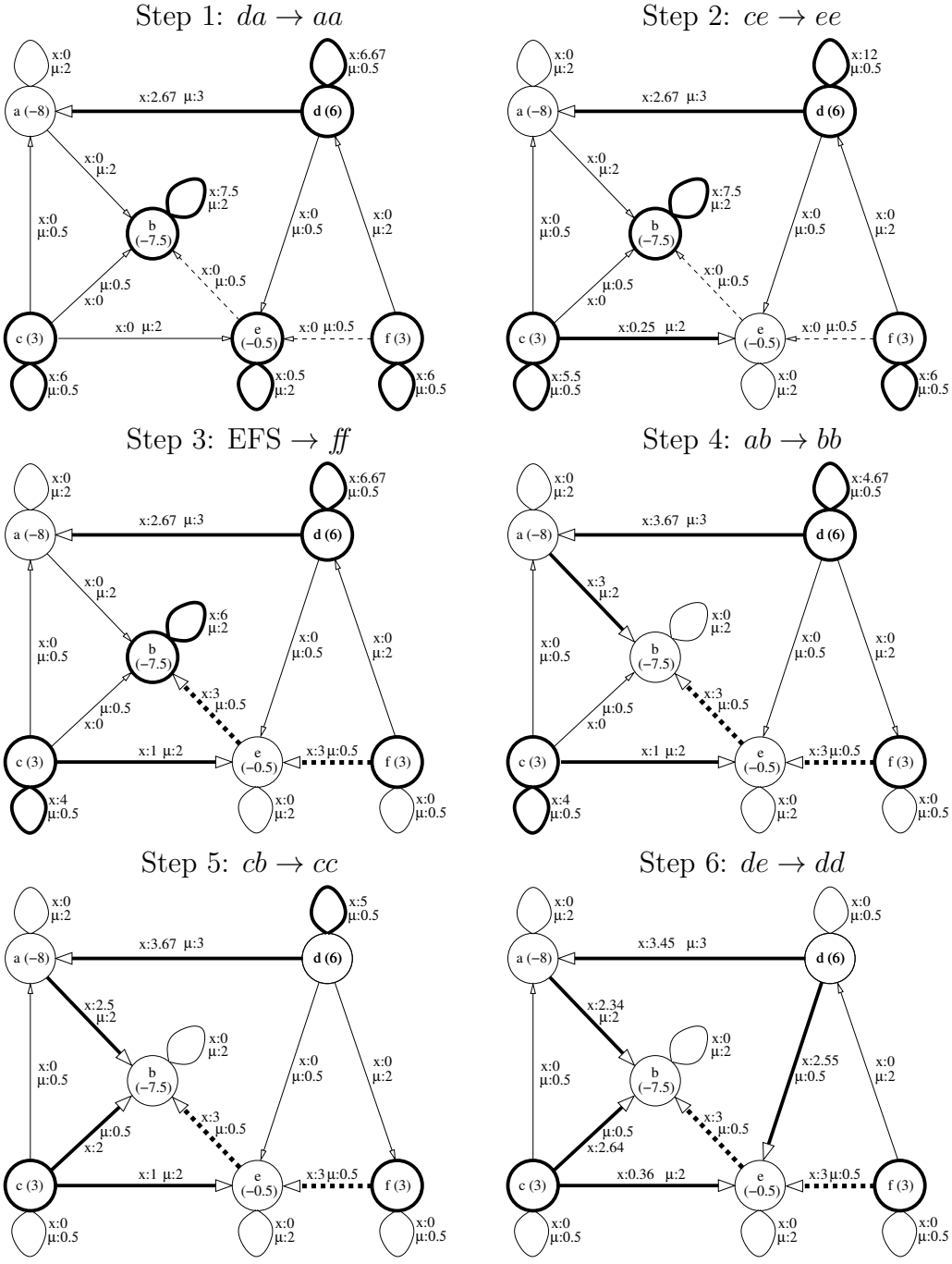

Figure 3: The steps taken by GenNetEq to determine the optimal flow for the network shown in figure 2. Bold arcs represent arcs in the basis, and bold nodes represent the roots of the trees in the basis. Dashed lines indicate equal flow sets. For each step $kl \to pq$ indicates kl enters the basis and pq leaves. After six steps, the algorithm terminates with an optimal cost of 86.70.

D. Pseudocode

Algorithm 1 Network Simplex for Generalized Networks with Equal Flow Sets

procedure GENNETEQ($G = (N, A), b, \mathcal{P}$)
    Initialize $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ by adding self-loops
    // Phase I:
    Set cost of all artificial arcs to 1, and all other arcs to 0
    $x \leftarrow$ SOLVE($(\mathbf{B}, \mathbf{L}, \mathbf{U})$)
    if any artificial arc has non-zero flow:
        return Problem is infeasible
    // Phase II:
    Restore original costs
    $x \leftarrow$ SOLVE($(\mathbf{B}, \mathbf{L}, \mathbf{U})$)
    return cost of x
end procedure

procedure SOLVE($(\mathbf{B}, \mathbf{L}, \mathbf{U})$)
    $x \leftarrow$ COMPUTEFLOWS($(\mathbf{B}, \mathbf{L}, \mathbf{U}), b$)
    while true:
        $\pi \leftarrow$ COMPUTEPOTENTIALS($(\mathbf{B}, \mathbf{L}, \mathbf{U})$)
        Identify entering variable x_e by computing reduced costs using π
        if entering variable exists:
            $x \leftarrow$ PIVOT($(\mathbf{B}, \mathbf{L}, \mathbf{U}), x, x_e$)
        else: // the current basis is optimal
            return x
end procedure

Algorithm 2 Computing Flows for a Basis Structure

procedure COMPUTEFLOWS($(\mathbf{B}, \mathbf{L}, \mathbf{U}), \hat{b}$)
    Set flow on members of $\mathbf{L}$ and $\mathbf{U}$ to their lower and upper bounds, respectively
    Adjust supply vector $\hat{b}$ appropriately based on flows for members of $\mathbf{L}$ and $\mathbf{U}$
    for $R_r \in \mathbf{B}$:
        $x_r \leftarrow \theta_r$
    for each type II tree $\overline{T_i}$:
        Traverse tree from leaves to root to determine flows on arcs in terms of θ_r's
    Solve system of equations induced by the roots of the type II trees
    for each arc (i, j) in the type II trees:
        Solve for x_{ij} using the values for the θ_r's
    Update supply vector at remaining nodes based on values of the θ_r's
    for each type I tree $T_i + (\alpha_i, \beta_i)$:
        Solve type I trees using procedure given by Ahuja et al. (1993).
    return x
end procedure

Algorithm 3 Computing Node Potentials

procedure COMPUTEPOTENTIALS($(\mathbf{B}, \mathbf{L}, \mathbf{U})$)
    for each type I tree $T_i + (\alpha_i, \beta_i)$:
        Use the procedure given by Ahuja et al. (1993) to determine π_j for all nodes in T_i
    for each type II tree $\overline{T_i}$:
        $\pi_{h_i} \leftarrow \theta_i$ // h_i is the root of $\overline{T_i}$
        for each node j in $\overline{T_i}$:
            Solve for π_j in terms of θ_i
    Solve the system of equations in (3) to determine numerical values for all θ_i's
    Use the values of θ_i to solve for π_i for all nodes i in the type II trees
    return π
end procedure

Algorithm 4 Pivoting

procedure PIVOT($(\mathbf{B}, \mathbf{L}, \mathbf{U}), x, x_e$)
    Let A_e be the column corresponding to x_e in the constraint matrix
    if $x_e \in \mathbf{L}$:
        $\hat{b} \leftarrow -A_e$
    else: // $x_e \in \mathbf{U}$
        $\hat{b} \leftarrow A_e$
    $\Delta_{x_\mathbf{B}} \leftarrow$ COMPUTEFLOWS($(\mathbf{B}, \mathbf{L} \cup \mathbf{U}, \emptyset), \hat{b}$)
    for each $var \in \mathbf{B}$:
        if $\Delta_{var} > 0$:
            $\delta_{var} \leftarrow (u_{var} - x_{var})/\Delta_{var}$
        else if $\Delta_{var} < 0$:
            $\delta_{var} \leftarrow x_{var}/(-\Delta_{var})$
        else: // $\Delta_{var} = 0$
            $\delta_{var} \leftarrow \infty$
    $\delta_e \leftarrow u_e$
    $\delta \leftarrow \min_{var \in \mathbf{B} \cup \{x_e\}} \{\delta_{var}\}$
    $x_l \leftarrow$ element from $\arg\min_{var \in \mathbf{B} \cup \{x_e\}} \{\delta_{var}\}$
    $x \leftarrow x + \delta \Delta_{x_\mathbf{B}}$
    if $x_e \in \mathbf{L}$:
        $x_e \leftarrow \delta$
    else: // $x_e \in \mathbf{U}$
        $x_e \leftarrow u_e - \delta$
    $\mathbf{B} \leftarrow \mathbf{B} \cup \{x_e\} - \{x_l\}$
    Update trees and indices as needed
    Add x_l to $\mathbf{L}$ or $\mathbf{U}$ appropriately
    return x
end procedure

Acknowledgments

The computational results reported were obtained using the Simulation and Optimization Laboratory. This research has been supported in part by the Air Force Office of Scientific Research (FA9550-10-1-0387), as well as the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. The authors would like to thank Dr. Hemanshu Kaul for work previously done on this problem. The authors would also like to thank Dr. Karen Aardal, the Associate Editor,

and two anonymous Referees for encouraging additional research that led to the significantly improved computational performance of the algorithm, as well as substantial improvements to the paper. This paper was chosen for the Best Team Research Project in the 2011 University of Illinois Department of Computer Science Graduate Research Symposium sponsored by Facebook.

References

Ahuja, R.K., T.L. Magnanti, J.B. Orlin. 1993. *Network Flows*. Prentice Hall, New Jersey.

Ahuja, R.K., J.B. Orlin, G.M. Sechi, P. Zuddas. 1999. Algorithms for the simple equal flow problem. *Management Sci.* **45** 1440–1455.

Ali, A.I., J. Kennington, B. Shetty. 1988. The equal flow problem. *Eur. J. Oper. Res.* **36** 107–115.

Armstrong, R.D., Z. Jin. 1997. A new strongly polynomial dual network simplex algorithm. *Math. Programming* **78** 131–148.

Bertsimas, D., J.N. Tsitsiklis. 1997. *Introduction to Linear Optimization*. Athena Scientific and Dynamic Ideas, LLC, Belmont, Massachusetts.

Bixby, R.E. 2002. Solving real-world linear programs: A decade and more of progress. *Oper. Res.* **50** 3–15.

Calvete, H.I. 2003. Network simplex algorithm for the general equal flow problem. *Eur. J. Oper. Res.* **150** 585–600.

Clark, R., L. Kennington, R.R. Meyer, M. Ramamurti. 1992. Generalized networks: Parallel algorithms and an empirical analysis. *ORSA J. Comput.* **4** 132–145.

Elam, J., F. Glover, D. Klingman. 1979. Strongly convergent primal simplex algorithm for generalized networks. *Math. Oper. Res.* **4** 39–59.

Klingman, D., A. Napier, J. Stutz. 1974. NETGEN: A program for generating large scale capacitated assignment, transportation and minimum cost flow networks. *Management Sci.* **20** 814–820.

Meyers, C.A., A.S. Schulz. 2009. Integer equal flows. *Oper. Res. Lett.* **37** 245–249.

Venkateshan, P., K. Mathur, R.H. Ballou. 2008. An efficient generalized network-simplex-based algorithm for manufacturing network flows. *J. Combin. Optim.* **15** 315–341.