Clustering for Faster Network Simplex Pivots

David Eppstein
Department of Information and Computer Science, University of California, Irvine, California 92697-3425

We show how to use a combination of tree-clustering techniques and computational geometry to improve the time bounds for optimal pivot selection in the primal network simplex algorithm for minimum-cost flow and related problems and for pivot execution in the dual network simplex algorithm, from $O(m)$ to $O(\sqrt{m})$ per pivot. Our techniques can also speed up network simplex algorithms for generalized flow, shortest paths with negative edges, maximum flow, the assignment problem, and the transshipment problem. © 2000 John Wiley & Sons, Inc.

Keywords: network simplex; minimum-cost flow; pivot rules

1. INTRODUCTION

Many of the problems in graph and network optimization, including shortest paths, minimum spanning trees, and matchings, are subsumed in the topic of network flows. The literature on this subject is enormous, comprising many books and hundreds of journal articles (for a recent survey on network flow, see the comprehensive text by Ahuja et al. [1]).

A practical and commonly used method for solving network flow problems is provided by the *network simplex algorithm* [3]. There are two important factors that contribute to the time taken by the network simplex method: the number of *pivots* and the time per pivot. There has been much research on choosing a *pivot rule* to produce few pivots, but less on reducing the time per pivot. In this paper, we show that a clustering technique developed by Frederickson [8, 9] and applied in many dynamic graph problems [7–9, 13] can be used to speed up a number of standard pivot rules for the network simplex algorithm.

Our results give the first sublinear time bound for pivot selection in the network simplex algorithm. They also provide the first case in which the clustering techniques that we use have been applied to a directed graph problem.[1] Some of our results involve a novel combination of persistent range query data structures from computational geometry with graph theoretic techniques related to clustering.

1.1. Network Simplex Methods

The *network simplex algorithm* [3] is a specialization of the simplex method from linear programming, as applied to network problems such as minimum-cost flow. Polynomial bounds are known for several variants of the network simplex algorithm (e.g., see [10, 12]). Even though these bounds are generally asymptotically larger than those of the known combinatorial algorithms (or of interior point linear programming algorithms [16]), network simplex remains an important practical alternative [1, 12], apparently because its running time, in practice, is smaller than the theoretical bound.

As in the standard linear programming simplex algorithm, network simplex maintains a feasible solution known as a *basis*, which is gradually improved in small steps known as *pivots* until optimality is reached. For the minimum-cost flow problem, the basis is a flow satisfying all capacity constraints and having the following treelike structure: The arcs of the network are partitioned into three subsets (T, L, U). No flow passes across any arc in L. Each arc in U carries flow exactly equal to its capacity. The remaining arcs, in T, form an (undirected) spanning forest of the network. The flows on these arcs are within the capacities of the arcs, but are not necessarily at the upper or lower limits of those capacities. Since T has no circuits, the flows on its arcs can be uniquely determined from the partition (T, L, U). A *pivot* consists of finding a negative-cost circuit induced in T by an arc a in L or U and passing flow around that circuit until some arc is saturated and no more flow is possible. The saturated arc is placed either in L or U and a is added to T. If there are multiple saturated arcs, an appropriate tie-breaking rule is used. This results in a new flow of the same form, and the process repeats.

It is a fundamental theorem of the network simplex method that for any minimum-cost flow problem there is an optimal solution having the tree structure described

Received November 1994; accepted November 1999
Correspondence to: D. Eppstein; e-mail: eppstein@ics.uci.edu
Contract grant sponsor: NSF; contract grant number: CCR-9258355

and that the method described above will always terminate with an optimal solution.

The time taken by the network simplex method is simply the number of pivots made, multiplied by the average time per pivot. However, at each step, there may be many possible choices of pivots, and the total number of pivots eventually made will depend to a large extent on how these choices are made. Many pivoting rules have been suggested, but one of the best in terms of its effect on the total number of pivots [1] is also one of the simplest and earliest. *Dantzig's pivot rule* chooses the induced circuit with the most negative total cost, ignoring the amount of flow that can pass through it.

Unfortunately, known implementations of Dantzig's rule test all induced circuits every time a pivot is chosen. We can test each circuit in constant time by using some simple vertex indices to measure the costs of paths in T. But this still results in $O(m)$ time per pivot and (even though the rule makes few pivots) a slow overall algorithm.

Instead, implementations of the network simplex algorithm have resorted to simpler pivot rules (such as cyclically scanning the edges not in T) which tend to find pivots more quickly. For some such rules the known bounds on time and pivots may be within $o(m)$ of each other. However, this does not truly bound the average time per pivot, so if we are using network simplex with the hope that the number of pivots will often be smaller than the theoretical bound, such results are not useful. Instead, we wish to show that a good pivot selection rule such as Dantzig's rule can be implemented in small average time per pivot, no matter how few pivots are actually made. This paper provides the first such results.

1.2. New Results

In this paper, we show that clustering can be used to speed up Dantzig's rule for the network simplex algorithm. We describe a method of implementing Dantzig's rule for which the time per pivot is $O(\sqrt{m})$, improving the previous $O(m)$ bound. To our knowledge, this is the first sublinear time bound for any form of pivot selection in the network simplex algorithm.

Our technique extends to other pivot rules, including the *candidate list* method which generalizes both cyclic scanning and Dantzig's rule. Clearly, our technique also applies to applications of the network simplex method for special cases of minimum-cost flow, including shortest path trees, maximum flows, the assignment problem, and the transshipment problem.

We describe an extension of our technique to the *dual network simplex method*, applied by Plotkin and Tardos in a strongly polynomial algorithm for the transshipment problem and for minimum-cost flows [12]. In the dual simplex method, the basis is a flow that is in some sense optimal but may not be feasible, again having a treelike structure. Each pivot reduces the flow on some arc to the capacity of that arc. For this method, the infeasible arcs on which to pivot are easier to find and pivot selection is less of a problem. The difficulty is now finding the optimal circuit containing a tree arc, in which to perform the desired pivot. For this problem, also, clustering can provide an improvement, from $O(m)$ to $O(\sqrt{m})$ time per pivot. We thus reduce Plotkin and Tardos' bounds of $O(m^2 n \log n)$ for the transshipment problem and $O(m^3 \log n)$ for the min-cost flow problem to $O(m^{3/2} n \log n)$ and $O(m^{5/2} \log n)$, respectively.

We finally discuss an even more inclusive network simplex problem: generalized flow. The bases now consist of pseudoforests rather than trees, but clustering can still apply to such structures. However, it is less easy to use clustering to determine the optimal pivot or even to find a possible pivot. We describe a method that combines clustering with a geometric convex hull algorithm to achieve bounds of $O(m^{2/3} \log n)$ per pivot, with a pivot selection rule that appropriately generalizes Dantzig's rule. A similar bound holds for selecting the pivot that achieves the maximum total decrease in overall cost, both for minimum-cost flow and for generalized flow.

2. CLUSTERING

Our algorithms use a technique of partitioning trees and forests into smaller subtrees or clusters of vertices introduced by Frederickson [8, 9].

This technique allows the trees to be modified by insertion or deletion of edges; in our application, we combine these updates into a *swap*, the replacement of an edge by some other edge, preserving the property that the resulting graph is a tree. Each pivot in the network simplex algorithm corresponds to a swap in the tree representing the algorithm's current basis.

We first transform our input graph G into a new graph G' with degree at most three, so any tree in G' will be binary. Let v be any edge of degree $\Delta > 3$; replace v by $\Delta - 2$ vertices connected by a path. Path endpoints receive two of the original edges of v, and each interior vertex receives one. Path edges have infinite bidirectional capacity and zero cost. Flows in G' correspond one-for-one with flows in G. All path edges will be included in T when we begin the simplex algorithm; they can never be saturated and so remain in T. Any network simplex algorithm on G' can be interpreted as a network simplex algorithm on G, and we can use our data structure on G' to choose and perform pivots for a network simplex algorithm running directly on G.

The following definitions are due to Frederickson [9], although our wording is slightly different:

Definition 1. *For any set S of vertices in a tree T, we define the degree of S to be the number of edges of T having precisely one endpoint in S.*

Definition 2. *A restricted partition of order z with respect to a binary tree T is a partition of the vertices of V such that*

1. *Each set in the partition contains at most z vertices.*
2. *Each set in the partition induces a connected subtree of T.*
3. *For each set S in the partition, if S contains more than one vertex, then the degree of S is at most two.*
4. *No two sets can be combined and still satisfy the other conditions.*

Such a partition can easily be found in linear time. There are at most $O(n/z)$ sets in a restricted partition of an n-vertex tree. If we change the tree by performing a swap, we can update the restricted partition in time $O(z)$ by splitting and remerging $O(1)$ partition sets [9].

Definition 3. *A restricted multilevel partition of order z is a sequence of partitions having the following properties*:

1. *The finest partition in the sequence is a restricted partition of order z; its partition sets are known as* basic clusters.
2. *Each successive partition is a restricted partition of order* 2 *of the tree formed by contracting the clusters in the previous partition.*
3. *In the coarsest partition, a single cluster contains all vertices.*

A restricted multilevel partition has $O(\log n)$ levels. Each cluster in the partition is divided into at most two clusters in the next lower level of the partition, so we can define a binary *topology tree* giving the inclusion relations between clusters on adjacent levels. For any swap in the tree, we can update the multilevel partition and its topology tree by updating the basic clusters and then merging and splitting $O(1)$ clusters at each higher level, in total time $O(z + \log n)$ [9].

Definition 4. *A* 2*-dimensional topology tree* (*for a given multilevel partition*) *consists of a node for every pair of clusters at the same level of a restricted multilevel partition. The parent of a node formed by two clusters A and B at a given level is defined to be the node at the next higher level, formed by the two clusters at that higher level that contain A and B. The two clusters forming a node need not be distinct; the root of the* 2*-dimensional topology tree is the node* (T, T).

The 2-dimensional topology tree can be updated in $O(z + m/z)$ time per change in the underlying tree, since the partition itself takes $O(z + \log n)$ time and since $O(m/z)$ nodes in the 2-dimensional topology tree must be updated [9]. We will typically balance the two portions of the bound by letting $z = \sqrt{m}$ to achieve an overall $O(\sqrt{m})$ bound.

3. DANTZIG'S RULE

Dantzig's rule, in brief, selects an edge in a tree-structured flow (T, L, U) inducing a cycle of minimum cost. To make this precise, we first define the cost of a path or cycle in G. We consider a path or cycle to be oriented in one consistent direction, but the arcs in it may be individually oriented either with or against the total direction of the path or cycle. The cost of the path or cycle is then simply the sum of the costs of the edges oriented with it, minus the sum of the costs of the edges oriented against it. In other words, this measures the total cost of pushing a single unit of flow along the path or around the cycle.

Given a tree-structured flow (T, L, U), there is a unique path in T connecting any pair of vertices, and any edge e in L or U induces a unique cycle with T, which we orient to match the orientation of e if e is in L and against the orientation of e if e is in U. In other words, the orientation is chosen so that it is possible to push more flow in G around the cycle in the chosen direction.

Dantzig's rule, then, selects that edge $e \in L \cup U$ for which the directed cost of the circuit induced by e in T, in the direction along which e is not already saturated, is as negative as possible.

We now describe our technique for implementing this pivot rule efficiently.

We keep a restricted multilevel partition of T and a corresponding 2-dimensional topology tree. For each nontrivial cluster of degree two at each level of the partition, we store the costs of the paths between the two vertices by which that cluster is connected to the rest of T. We choose one of the two vertices as a *reference point* for the cluster. In a cluster of degree one, we choose the single vertex connecting that cluster to the rest of the graph as its reference point. For each edge e in $L \cup U$, we keep track of the distances from the endpoints of e to the reference points in the two basic clusters containing them.

The 2-dimensional topology tree for T contains a node $N_{\alpha\beta}$ for each pair of clusters α and β in a given level of the partition. We now describe the information stored at these nodes.

First consider the case that α and β differ. Then, at node $N_{\alpha\beta}$, we store a single arc chosen among all arcs in L directed from a vertex in α to a vertex in β. For any such arc e, we can form a path by concatenating the path in α from the reference point to one endpoint of e, together with e itself and the path in β from the endpoint of e to the reference point of β. The arc that we store is the one minimizing the length of this path. Similarly, we store another arc chosen among all arcs directed the other way, from β to α, minimizing the cost of the path connecting the reference points of the clusters via that arc. For both arcs, we also store the cost of the corresponding path. In each case, the arc that we store must clearly also have been chosen at the next lower

level of the topology tree, so there are $O(1)$ candidate arcs to chose from at each nonbasic level of the topology tree. For each candidate, we can compute the path cost in constant time, by combining a similar cost at the next lower level with $O(1)$ distances between lower-level cluster endpoints. Therefore, recomputing the information stored at $N_{\alpha\beta}$ can be performed in constant time.

In the other case, $\alpha = \beta$. Then, at node $N_{\alpha\alpha}$, we choose a single arc, chosen among all arcs with both endpoints in α and inducing the minimum-cost circuit among such arcs, and store with it the cost of the circuit. If α is not a basic cluster, denote the children of α in the topology tree by γ and δ. Then, either the minimum-cost circuit is entirely contained in one of γ and δ or it has one endpoint in each cluster. In the first case, the chosen arc will also have been chosen in $N_{\gamma\gamma}$ or $N_{\delta\delta}$, and we will already have available the length of the induced circuit. In the second case, the arc must have been one of the two chosen in $N_{\gamma\delta}$; we can compute the length of the induced circuit by combining the lengths of the paths stored with the arcs in $N_{\gamma\delta}$ with the paths between reference points in γ and δ and the edge connecting the two clusters. Thus, we can again choose the arc in constant time from the information stored at the next lower level.

The pivot chosen by Dantzig's rule is then simply the arc chosen by the node at the root of the 2-dimensional topology tree.

Theorem 1. *In the network simplex algorithm for minimum-cost flow, we can select a pivot according to Dantzig's rule, perform the pivot, and update our data structures for the next pivot, in $O(\sqrt{m})$ time.*

Proof. The pivot can be determined in constant time from the data structure that we described. We now describe how to perform the pivot. The first thing that must be done is to determine the saturated edge of T that is replaced by the pivot edge. This can be done in time $O(\log n)$ per pivot using the dynamic tree data structure of Sleator and Tarjan [10, 14, 15] or, alternatively, in time $O(m^{1/2})$ using our tree partition data structure.

Once we have determined the swap made in T by the pivot, we must update the data structures used to find the next pivot. As seen in the previous section, the restricted partition and topology tree take $O(\sqrt{m})$ time to update. As noted above, the arcs chosen at higher levels of the 2-dimensional topology tree can be updated after a change in time $O(1)$ per affected node, or $O(\sqrt{m})$ total. The information stored in each arc about the distances to basic node vertices can be updated in $O(\sqrt{m})$ time, as can the information stored in each basic cluster about distances between endpoints and about induced cycles contained in the basic cluster. The endpoint distance at each higher level can be computed in constant time. The total number of candidate edges among leaf nodes of the 2-dimensional topology tree is $O(\sqrt{m})$, since each such edge is in one of the $O(1)$ basic clusters affected by an update, so all such nodes can select their minimum-cost arcs in total time $O(\sqrt{m})$; this bound also holds for propagating these selections to higher levels of the tree. ∎

4. CANDIDATE LIST RULE

The *candidate list* pivot rule [1, 11] speeds up Dantzig's rule by reducing the number of edges from which selection must be made. It operates in *major* and *minor cycles*. For every major cycle, this algorithm selects a certain number k of candidate pivot edges in $L \cup U$. Then, the algorithm performs a series of minor cycles, each consisting of a pivot by the least-cost circuit induced by a candidate edge (as in Dantzig's rule). When no such pivots are possible, a new major cycle begins.

We ignore the cost of a major cycle, which is typically small, and concentrate on the time-consuming minor cycles. A naive implementation of this rule takes time $O(k + n)$ per minor cycle: $O(k)$ to select a pivot and $O(n)$ to update certain *tree indices* used to select the next pivot.

Theorem 2. *In the network simplex algorithm for minimum-cost flow, we can select and perform each pivot according to the candidate list rule, in $O(\sqrt{k})$ time per minor cycle.*

Proof. Within each major cycle, we can reduce the problem to one on a graph with $O(k)$ edges and vertices, by ignoring the noncandidate edges in $L \cup U$ and contracting certain edges in T. More specifically, at most $2k$ vertices in T are adjacent to candidate edges. So, T consists of at most $2k - 1$ paths between these vertices, together with a number of side trees not connecting any candidate vertices. All edges in such side trees, and all but the edge of smallest residual capacity in each direction of each path, will remain in T for the duration of the major cycle and can safely be contracted. In the contracted graph, the pivots performed are those of Dantzig's rule and so can be found in the stated time bound by applying the data structure of Theorem 1 to the contracted graph. ∎

The contraction step needed for this speedup can be performed in time $O(n+k)$ per major cycle, which would typically be dominated by the cost of finding the candidate list.

Although this $O(\sqrt{k})$ bound is the best theoretical guarantee we have been able to prove, it may be possible to use similar ideas to get better run times in practice. If we (heuristically) assume that $\Omega(k)$ pivots are performed per major cycle, a good choice would be to let $k = \Theta(m^{2/3})$. The average time to select candidates in each major cycle would be $O(m^{1/3})$ per pivot, balancing the time to perform pivots in each minor cycle. With similar assumptions, a pivot strategy with more than two levels of cycles could do even better. For ex-

ample, in each cycle, one might pick half the cycle's candidate edges as candidates for a lower-level cycle in a contracted graph, recursing until the graph has a constant size. We would expect this strategy to give logarithmic time per pivot in practice. However, we can prove no such result as a worst-case bound, so experiments would be needed to determine how well this iterated contraction method works. For a similar algorithm of repeated recursive selection and contraction (for a different problem, but with provable worst case time bounds), see [6].

5. DUAL NETWORK SIMPLEX

We next consider the dual network simplex algorithm, a version of which was shown by Plotkin and Tardos [12] to run in strongly polynomial time. In the minimum-cost flow dual simplex algorithm, a basis consists of a tree-structured flow which satisfies certain cost optimality conditions, but which may violate capacity constraints on arcs in T. An arc is *infeasible* if its capacity constraint is violated. A pivot consists of selecting an infeasible arc in T and making it feasible by pushing the flow around a cycle containing that arc. The cycle is chosen to be the one induced in T by a *replacement arc* in $L \cup U$ for which the total cycle cost is minimum.

Thus, duality reverses the roles of pivot selection and execution. Typical pivot selection rules (such as selecting the most infeasible arc) can be performed quickly with a dynamic tree data structure [14], but pivot execution requires finding an optimal arc in $L \cup U$. We now show how to speed up this pivot execution problem, again using tree-clustering techniques:

Theorem 3. *In the dual network simplex algorithm for minimum-cost flow, we can execute a given pivot and update our data structures for the next pivot, in* $O(\sqrt{m})$ *time.*

Proof. We again maintain a 2-dimensional topology tree of clusters in T. For each node in the topology tree, representing a pair of clusters in T, we keep two candidate replacement arcs, giving the minimum-cost paths in each direction between the two reference points of the clusters, and we also keep the costs of these two paths. To perform a pivot, we remove the pivot edge from T, resulting in a forest of two trees. We then split $O(1)$ clusters per level of the multilevel partition so that each endpoint of the pivot arc is alone in a trivial cluster. At the top level, there will be four clusters, the two trivial ones and two larger ones on each side of the cut formed by removing the pivot arc from T. We merge each trivial cluster with the corresponding larger cluster and update the 2-dimensional topology tree. The desired arc will be chosen by the node in the 2-dimensional topology tree corresponding to the remaining top-level pair of clusters. ∎

In particular, this applies in Plotkin and Tardos' [12] algorithm for the transshipment problem, which they also use to solve the minimum-cost flow problem. In this problem, there are flow demands at each vertex but no capacity constraints. The basis has a certain amount of flow on each tree arc and no flow on other arcs. Infeasible arcs are those for which the total flow demand across the corresponding cut has not been met. Again, some such arc is selected as a pivot and we execute the pivot by finding the minimum-cost induced cycle containing the pivot arc.

Theorem 4. *The dual network simplex algorithm of Plotkin and Tardos can be made to run in time* $O(m^{3/2} n \log n)$ *for the transshipment problem and* $O(m^{5/2} \log n)$ *for the minimum-cost flow problem.*

Proof. In Plotkin and Tardos' algorithm, a pivot arc must be selected in some subtree of T, but this is still straightforward with the dynamic tree data structure of Sleator and Tarjan. The bottleneck is pivot execution, which is the same with this rule as it is above. Thus, we can select and execute each pivot of their algorithm in time $O(\sqrt{m})$. The fact that the graph G' (formed by expanding vertices of G) has more vertices than has G does not affect Plotkin and Tardos' $O(mn \log n)$ bound on the number of pivots, as we can perform the dual network simplex algorithm in G directly and use the data structure in the larger graph G' only to determine the replacement arc in each pivot. ∎

6. GENERALIZED FLOW

We now return to pivot selection in the primal network simplex algorithm. We consider a more complicated flow problem, that of *generalized flow* in which the goal is to find a flow of minimum cost in a network for which each arc has three parameters: cost and capacity as before, but also a factor by which flow through the arc is multiplied. We can then define a *generalized flow* to be a set of nonnegative flow values on (the head of) each arc, bounded by the arc capacities, such that at any vertex the sum of flow values times multipliers on the incoming arcs equals the sum of unmultiplied flow values on the outgoing arcs. The cost of a generalized flow is simply the sum of the product of flow values by arc costs.

We use the network simplex formulation in [1]. A *basis* consists of a flow on arcs partitioned as before into three sets (T, L, U). L and U are as before but T now consists of arcs forming a *spanning pseudoforest*, a graph such that each component consists of a tree and a single additional edge inducing a cycle in the tree. We can expand the flow around such a cycle by pushing it one way and contract the flow by pushing it the other way.

A *pivot* consists of an edge in L or U. If the pivot connects two components of the pseudoforest, we can

generate flow in the cycle from one component and absorb the flow in the cycle from the other component until some edge is saturated. We add the pivot edge to T and remove the saturated edge, producing one or two components in a new pseudoforest. Similarly, if the pivot edge connects two endpoints in a single component, adding it to that component will produce two cycles, and we can pass flow from one to another until an edge is saturated. Removing the saturated edge again produces one or two pseudoforest components. The total cost improvement of the pivot is the cost to generate flow at one endpoint of the pivot arc, the cost of the arc itself, and the cost to absorb flow multiplied by the flow multiplier of the arc. We wish to find some arc for which this cost is negative or (as in Dantzig's rule) for which the cost is minimum.

The fact that we are using pseudoforests instead of trees is not a major concern for the clustering method, nor for the dynamic trees used to execute each pivot. We can remove one edge from each component, producing a tree which we represent as before, and maintain separately the information about the removed edge.

The difficulty with executing the network simplex algorithm for generalized flow stems instead from the fact that the choice of the optimal pivot arc connecting any two clusters cannot be determined from the part of the network contained in the two clusters, but depends also on the cost to generate flow in the first cluster and absorb flow in the second cluster (the parameters a and g below), which can be determined by parts of the pseudoforest far from the clusters themselves. Therefore, we cannot choose a single optimal arc per node in a 2-dimensional topology tree, as we did for the simpler minimum-cost flow problem.

Instead, we use a single-level partition of the pseudoforest, into $O(k)$ clusters of $O(m/k)$ vertices each, for a parameter k to be chosen later. There may also be a large number of pseudoforest components with $o(m/k)$ vertices each; these must be treated slightly differently. For each pair of clusters connected by at least one arc, we keep a data structure that can determine quickly the best pivot arc. After each update, we query all $O(k^2)$ such data structures to determine the overall best pivot.

For each arc connecting each pair of clusters we compute three values: first, the total cost c of the path through the arc connecting the reference points of the two clusters, measured per unit of flow into the arc; second, the flow multiplier μ_1 of the path from the first reference point to the arc; and, third, the flow multiplier μ_2 of the path through the arc to the second reference point. When we wish to determine the best pivot among all arcs connecting a given pair of clusters, we first determine the cost g of generating a unit of flow at the reference point of the first cluster and the cost a of absorbing a unit of flow at the reference point of the second cluster. The total cost of a pivot with parameters (c, μ_1, μ_2) is then $g/\mu_1 + c + a\mu_2$. This is a linear function of $c, 1/\mu_1$, and μ_2, so we can find the pivot with least total cost by finding an extremum of the convex hull of the points $(c, 1/\mu_1, \mu_2)$ in Euclidean space using a point location algorithm. Thus, we store with each cluster the convex hull of these points, represented in a way that lets us find its extrema quickly.

To handle the possibility of many small pseudoforest components, we also group these into clusters. For each cluster, we introduce an artificial reference point for which the cost of generating or absorbing flow is unity, connected to a point in each component by a zero-cost edge with an appropriate flow multiplication factor.

Theorem 5. *In the network simplex algorithm for generalized flow, we can select the minimum-cost pivot, perform the pivot, and update our data structures for the next pivot, in $O(m^{2/3} \log n)$ time. For planar networks, the time is $O(m^{1/2} \log n)$ per pivot.*

Proof. The clustering itself can be updated in $O(m/k)$ time. Each update causes a constant number of merges and splits, among clusters with a total of $O(m/k)$ arcs. When we perform an update, we must also recompute the 3-dimensional points representing arcs incident to changed clusters and the convex hulls of these points for the cluster pairs in which one of the two clusters has been changed. Since these cluster pairs involve a total of $O(m/k)$ arcs, the corresponding hulls involve a total of $O(m/k)$ points. The points themselves can be recomputed in $O(m/k)$ time, and their hulls can be found in a total time of $O((m/k) \log n)$.

After updating the clustering and the associated convex hull structures, we must compute the new best pivot by finding the extrema of linear functions in $O(k^2)$ hulls. For planar networks, only $O(k)$ cluster pairs share edges so we need only find extrema in $O(k)$ hulls. Whenever we recompute a convex hull, we can also build a point location data structure that allows these queries to be answered in logarithmic time each [4]; therefore, the total time for this step is $O(k^2 \log m)$ [$O(k \log m)$ for planar networks]. Pivot execution takes $O(\log n)$ time with a dynamic tree data structure. Letting $k = m^{1/3}$ ($k = m^{1/2}$ for planar networks) results in the stated bounds. ∎

A modest further improvement is possible if we use a multilevel partition, in which the top-level clustering has $O(k)$ clusters with $O(m/k)$ vertices each and each successive level has clusters formed by splitting the clusters of the next higher level in two. (Note that this is not quite the same as a restricted multilevel partition because the top level has more than one cluster.) Again, we maintain the convex hull of a set of 3-dimensional points associated with the arcs connecting each cluster pair in each level. The hulls associated with the top-level clusters are used as before to find pivots; the hulls at lower levels are used only to update the larger hulls. Then, whenever we need to recompute a convex hull for

a cluster pair, it can be found as the hull of the union of at most four hulls associated with smaller cluster pairs. The hull of the union of two convex hulls is the projective dual of the intersection of two convex polyhedra and can be found in linear time by Chazelle's algorithm [2], so the total time for recomputing convex hulls at all levels of this data structure is $O(m/k)$ per operation. Setting $k = m^{1/3}\log^{-1/3} n$ ($k = m^{1/2}\log^{-1/2} n$ for planar networks) gives time bounds of $O(m^{2/3}\log^{1/3} n)$ [$O(m^{1/2}\log^{1/2} n)$, respectively] per operation. We omit further details since the improvement is small.

A somewhat simpler algorithm with planar convex hulls can be used if we measure pivot costs per unit flow at the clusters' reference points.

7. GREEDY PIVOT RULE

In previous sections, we chose a pivot with minimum cost per unit flow. Although this works well for minimum-cost flow, we had some definitional difficulty with generalized flow because the amount of flow in a given pivot differs from edge to edge of the network.

As an alternate pivot selection rule, we now consider the *greedy pivot rule*: Find the pivot with minimum total cost, taking into account the amount of flow. For most of this section, we will discuss the greedy pivot rule for standard minimum-cost network flow; we mention generalized flow again only briefly at the end of the section.

As with the methods of the previous section, the best pivot connecting a pair of clusters will depend on factors external to the cluster. We will again use a single-level partition, with data structures for each cluster pair which we can query to determine the best pivot involving that pair. If there are k clusters, each update will cause the reconstruction of data structures for a total of $O(m/k)$ arcs, and each pivot can be found by querying $O(k^2)$ data structures [$O(k)$ for planar networks]. As before, our overall time bound will be found by balancing these two terms.

We use the *ambivalent data structure* technique of Frederickson [9]: Rather than keeping a single data structure for each pair of clusters, as in the previous section, we keep at most four data structures per pair. By the definition of a restricted partition, each cluster has at most two *reference points*, which we define to be the endpoints of arcs having exactly one endpoint in the cluster. If a path in the tree starts at a vertex of one cluster and ends at a vertex of a different cluster, it must pass through at least one reference point in each cluster. For each pair of clusters, we store a data structure for each choice of one of the reference points from each of the two clusters.

Consider a potential pivot by some given arc a, connecting a pair of clusters. We partition the edges involved in the pivot into two sets: The first set consists of edges within the two clusters and a itself, and the second set consists of all remaining vertices. Then, these two sets form two paths, connected to each other at reference points of the two clusters. Let c_i be the cost of the path through arc a ending at the two reference points, and let r_i be the minimum remaining capacity on any edge in that path. Let c_x be the cost of the path in T connecting the two cluster reference points, and let r_x be the minimum remaining capacity on any edge in that path. Then, the total cost of the pivot is simply $(c_i + c_x)\min(r_i, r_x)$. We maintain a data structure that can find the pivot minimizing this value, among all arcs in $L \cup U$ connecting the two clusters. The values of c_i and r_i will be known when we construct the data structure, and the values of c_x and r_x will be known when we query it.

We find the minimum value of $(c_i + c_x)\min(r_i, r_x)$ in three steps: First, we determine those arcs for which $r_i \geq r_x$; for such arcs, we must merely minimize c_i. Second, among the remaining arcs, we minimize $(c_i + c_x)r_i$. This is a linear function of $c_i r_i$ and of r_i alone, so it can be minimized by a binary search on the convex hull of the planar point set formed by taking a point $(c_i r_i, r_i)$ for each potential pivot arc. Third, we compare the two values found in the first two steps and choose the smaller of the two. We can perform these steps with a data structure formed as follows: Consider adding the points $(c_i r_i, r_i)$ to a convex hull data structure, one at a time, point i at time $t = r_i$, and simultaneously removing the points from a priority queue keyed by value of c_i. If we could recover the data structure as it existed at time $t = r_x$, we could use the priority queue to perform the first step and the convex hull to perform the second step. But this recovery in the data structure history can be done using *persistent data structure* techniques [5]. Thus, we can build in time $O(n\log n)$ a data structure that can minimize $(c_i + c_x)\min(r_i, r_x)$ for a single cluster pair (connected by n arcs) in $O(\log n)$ time per query.

Theorem 6. *In the network simplex algorithm for minimum-cost flow, we can select a pivot according to the greedy rule, perform the pivot, and update our data structures for the next pivot, in $O(m^{2/3}\log n)$ time. For planar networks, the time is $O(m^{1/2}\log n)$ per pivot.*

Proof. As in Theorem 5, we form a single-level clustering with $O(k)$ clusters of $O(m/k)$ vertices. Each update involves the splitting and merging of $O(1)$ clusters and the rebuilding of the associated persistent hull data structures for a set of cluster pairs involving a total of $O(m/k)$ arcs, in time $O((m/k)\log n)$.

To find the best pivot, we then test all $O(k^2)$ cluster pairs [$O(k)$ pairs for planar networks]. For each pair, we determine the two relevant reference points and query the data structure associated with that pair of points, in time $O(\log n)$. Letting $k = m^{1/3}$ ($k = m^{1/2}$ for planar networks) results in the stated bounds. ∎

For generalized flows, the situation is more complicated: The cost of a pivot is a function of the costs of the edges within the cluster, the multiplication factors of

those edges, the residual capacities of those edges, and several external factors. The capacity may be limited by edges external to the cluster, by edges within the cluster but not on a flow-generating cycle, or by edges on a flow-generating cycle. One must consider separate cases for pivots connecting two components of the pseudoforest and pivots connecting two points in the same component. The possibility of many small components again gives rise to further difficulty. It seems likely that the type of capacity-limiting edge can be determined in polylogarithmic time by orthogonal range-searching techniques, after which the total cost could be computed using three-dimensional convex hull techniques applied to an appropriate formula linear in at most three terms specific to the pivot and depending on a number of other terms which are fixed per cluster pair. If so, it seems one should be able to compute greedy generalized flow pivots in time $O(m^{2/3} \log^{O(1)} n)$. However, we have not worked out the details of such an algorithm.

Notes

1. Perhaps it is worth noting that another major technique from dynamic graph algorithms, sparsification [7], does not appear to apply to these directed graphs. Thus, our time bounds have terms like $O(\sqrt{m})$, where the corresponding bounds for the dynamic graph algorithms cited above generally have terms like $O(\sqrt{n})$.

REFERENCES

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, Network flows: Theory, algorithms, and applications, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] B. Chazelle, An optimal algorithm for intersecting three-dimensional convex polyhedra, SIAM J Comput 21 (1992), 671–696.

[3] G.B. Dantzig, "Application of the simplex method to a transportation problem, " Activity analysis and production and allocation, Wiley, New York, 1951.

[4] D.P. Dobkin and D.G. Kirkpatrick, Fast detection of polyhedral intersection, Theor Comput Sci 27 (1983), 241–253.

[5] J.R. Driscoll, N. Sarnak, D.D. Sleator, and R.E. Tarjan, Making data structures persistent, J Comput Syst Sci 38 (1989), 86–124.

[6] D. Eppstein, Offline algorithms for dynamic minimum spanning tree problems, J Alg 17 (1994), 237–250.

[7] D. Eppstein, Z. Galil, G.F. Italiano, and A. Nissenzweig, Sparsification—A technique for speeding up dynamic graph algorithms, J ACM 44 (1997), 669–696.

[8] G.N. Frederickson, Data structures for on-line updating of minimum spanning trees, with applications, SIAM J Comput 14 (1985), 781–798.

[9] G.N. Frederickson, Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees, SIAM J Comput 26 (1997), 484–538.

[10] A.V. Goldberg, M.D. Grigoriadis, and R.E. Tarjan, Use of dynamic trees in a network simplex algorithm for the maximum flow problem, Math Prog 50 (1991), 277–290.

[11] J.M. Mulvey, Pivot strategies for primal-simplex network codes, J ACM 25 (1978), 266–270.

[12] S.A. Plotkin and É. Tardos, Improved dual network simplex, Proc 1st ACM/SIAM Symp Discrete Algorithms, 1990, pp. 367–376.

[13] M.H. Rauch, Fully dynamic biconnectivity in graphs, Algorithmica 13 (1995), 503–538.

[14] D.D. Sleator and R.E. Tarjan, A data structure for dynamic trees, J Comput Syst Sci 26 (1983), 362–391.

[15] R.E. Tarjan, Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm, Math Prog 78 (1997), 169–177.

[16] P.M. Vaidya, Speeding up linear programming using fast matrix multiplication, Proc 30th IEEE Symp Foundations of Computer Science, 1989, pp. 332–337.