Clustering for Faster Network Simplex Pivots

David Eppstein*

Department of Information and Computer Science
University of California, Irvine, CA 92717

Tech. Report 93-19

April 15, 1993

Abstract

We show how to use tree clustering techniques to improve the time bounds for optimal pivot selection in the primal network simplex algorithm for minimum cost flow, and for pivot execution in the dual network simplex algorithm for the same problem, 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.

*Work supported in part by NSF grant CCR-9258355.

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, Magnanti, and Orlin [1].

Although polynomial time combinatorial algorithms are known for many types of flow problem, a commonly used alternate method for solving these problems is provided by the *network simplex algorithm* [2], a specialization of the simplex method from linear programming. Polynomial bounds are known for variants of this algorithm (e.g. see [8, 10]) but these bounds are generally asymptotically larger than those of the combinatorial algorithms (or of interior point linear programming algorithms [13]). Nevertheless network simplex remains an important practical alternative [1, 10], 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 tree-like 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 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, 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 pivots to make, and the number of pivots will

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 pseudo-forests 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 $O(n^{2/3} \log n)$ 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 into smaller subtrees, or clusters of vertices introduced by Frederickson [6, 7].

We first transform the graph G into a new graph G' with degree at most three, so any tree in G' is 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.

Definition 1 (Frederickson [7]). *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 there are at most two tree edges having one endpoint in S.*
4. *No two sets can be combined and still satisfy the other conditions.*

depend to a large extent on these choices. 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 implementations of Dantzig's rule have been forced to 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 (although the worst case bound per pivot may still be $O(m)$), making up for the increase in the number of pivots.

In this paper we show that a clustering technique developed by Frederickson [6, 7] and applied in a variety of dynamic graph problems [5, 6, 7, 11] 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 pivot selection in the network simplex algorithm. For some pivot rules known bounds on total time and on pivots were within $o(m)$ of each other, but we wish to instead bound the time per pivot so that if fewer than expected pivots are made the improvement will still exist. Perhaps with our improvement Dantzig's rule will again be competitive with other network simplex methods.

Our technique extends to other pivot rules, including the *candidate list* method which generalizes both cyclic scanning and Dantzig's rule. Clearly it also applies to applications of the network simplex method in 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 [10]. In the dual simplex method the basis is a flow that is in some sense optimal but may not be feasible, again having a tree-like 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

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 removing an edge and adding a new one, we can update the restricted partition in time $O(z)$ by splitting and re-merging $O(1)$ partition sets [7].

Definition 2 (Frederickson [7]). *A restricted multi-level 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 multi-level 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 multi-level 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 $O(z + \log n)$ [7].

Definition 3 (Frederickson [7]). *A 2-dimensional topology tree consists of a node for every pair of clusters at the same level of a restricted multi-level partition. The children of each node are pairs of clusters at the next lower level that are contained in the clusters corresponding to the node.*

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 [7]. 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

Given a tree-structured flow (T, L, U), Dantzig's rule for the network simplex minimum cost flow algorithm 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. If e is in L we look for the cost of augmenting flow through e and then back from its head to its tail along the path in T connecting them. If e is in U we look for the cost of augmenting flow backwards through e and forwards through the path in T.

We keep a restricted multi-level partition of T, and a corresponding two-dimensional topology tree. For each cluster at each level of the partition we store the distance (directed sum of edge costs) between the two possible points by which it can connect to the rest of T. We choose one such endpoint as a *reference point* for the cluster For each edge in $L \cup U$, we keep track of the distance to the reference point in the two basic clusters it connects.

The two-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. First consider the case that α and β differ. For each such node we store a single arc chosen among all arcs in L connecting from α to β, and among all arcs in U connecting from β to α. Among all such arcs, we choose the one for minimizing the total cost of the path through that arc connecting the reference points of the clusters, and store along with that arc the cost of the path. Such an arc 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 non-basic level of the topology tree. For each candidates 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.

In the other case, $\alpha = \beta$. We then choose a single arc, inducing the minimum cost circuit within the cluster, and store with it the cost of the circuit. Either this circuit is entirely contained in one of the clusters at the next lower level, or the chosen arc connects two such clusters. In either case 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 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 [8, 12], 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. 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. 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, 9] speeds up Dantzig's rule by reducing the number of edges that must be selected among. It operates in *major* and *minor cycles*. Every major cycle, a certain number k of candidate pivot edges in $L \cup U$ are selected. Then the algorithm performs a series of minor cycles, each consisting of a pivot by the least cost circuit induced by a candidate edge. When no such pivots are possible, a new major cycle begins.

We ignore the cost of a major cycle, and concentrate on the 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 the tree indices used to select the next pivot. This can be sped up further, to $O(k)$, by observing that, within each major cycle, we can reduce the problem to one on a graph with $O(k)$ edges and vertices, by ignoring the non-candidate 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 algorithm and can safely be contracted. This contraction can be performed in time $O(n + k)$ per major cycle, which would typically

be dominated by the cost of finding the candidate list.

In the contracted graph, the pivots performed are those of Dantzig's rule or a slight modification thereof, and can be performed naively in $O(k)$. But our algorithm of Theorem 1 applies to give the following improvement:

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.*

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 example, 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 constant size. For a similar algorithm of repeated recursive selection and contraction (for a different problem, but with provable worst case time bounds) see [4].

5 Dual Network Simplex

We next consider the dual network simplex algorithm, a version of which was shown by Plotkin and Tardos [10] 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, and making it feasible by pushing flow around a cycle containing that arc. The cycle is chosen to be induced by a *replacement arc* for which the total cycle cost is minimum.

Thus duality reverses the roles of pivot selection and execution. Pivot selection (e.g. of the most infeasible arc) can be performed quickly with a dynamic tree data structure [12], but pivot execution requires finding an optimal arc in $L \cup U$.

We use essentially the same data structure as that for Dantzig's rule. For each pair of cluster endpoints we keep the candidate replacement arc that would be optimal if the pivot arc were on a path in T connecting those two endpoints, together with the cost of the portion of the induced cycle that

would be within the two clusters. To perform a pivot, we split $O(1)$ clusters per level of the multi-level 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.

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.*

In particular this applies in Plotkin and Tardos' [10] 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. In Plotkin and Tardos' algorithm, a pivot arc must be selected in some subtree of T, but this is still straightforward with a dynamic tree data structure. Thus we can select and execute each pivot of their algorithm in time $O(\sqrt{m})$. The fact that G' has more vertices than 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 using the data structure in 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 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 pseudo-forest*, that is 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 another pseudoforest. Similarly if the pivot is in a single component, there will now be two cycles in that component, and we can pass flow from one to another until an edge is saturated. 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. The difficulty stems rather from the fact that the choice of the optimal pivot arc connecting any two clusters cannot be determined from information internal to the two clusters, but depends also on the cost to absorb flow in the second cluster, which can be determined by parts of the pseudoforest far from the cluster itself. Therefore we cannot choose a single optimal arc per node in a 2-dimensional topology tree.

Instead we use a single-level partition of the pseudo-forest, into $O(m^{1/3})$ clusters of $O(m^{2/3})$ vertices each. There may also be a number of components with fewer than $O(m^{2/3})$ vertices each; these must be treated slightly differently. For each pair of clusters, we keep a data structure that can determine quickly the best pivot arc. After each update we query all $O(m^{2/3})$ 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 at 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 flow at the reference point of the first cluster, and the cost a of absorbing 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.

To handle the possibility of many small pseudo-forest 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 4. *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.*

Proof: The clustering itself can be updated in $O(m^{2/3})$ time. After each update we must compute convex hulls in a collection of point sets with total cardinality $O(m^{2/3})$, and perform point locations in $O(m^{2/3})$ hulls. Pivot execution takes $O(\log n)$ time with a dynamic tree data structure. □

A 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. As with generalized flow, the best pivot connecting a pair of clusters will depend on factors external to the cluster, so we will again use a single-level partition. We use the *ambivalent data structure* technique of Frederickson [7]: Rather than keeping a data structure per cluster pair, as in the previous section, we instead keep a data structure for each pair of vertices connecting clusters to the rest of T, which will be used to determine the best arc whenever the path in T connecting the two clusters goes through those two vertices.

We start with the minimum cost flow network simplex algorithm. There are at most two vertices connecting each cluster to the rest of T, so there are four possible combinations of such vertices. We treat each combination

separately. Thus we can fix a *reference point* in each cluster through which the path to the other cluster is assumed to pass. Suppose we wish to pivot by some given arc, connecting a pair of clusters. Let c_i be the cost of the path through the arc, 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 clusters, 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 [3]. Thus we can build in time $O(n \log n)$ a data structure that can minimize $(c_i + c_x)\min(r_i, r_x)$ in $O(\log n)$ time per query.

Theorem 5. *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 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 pseudo-forest, 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.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[2] G. B. Dantzig. Application of the simplex method to a transportation problem. *Activity Analysis and Production and Allocation*, ed. T. C. Koopmans. Wiley, 1951.

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

[4] D. Eppstein. Offline algorithms for dynamic minimum spanning tree problems. *2nd Worksh. Algorithms and Data Structures.* Springer LNCS 519 (1991) 392–399.

[5] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification, a technique for speeding up dynamic graph algorithms. *33rd IEEE Symp. Foundations of Computer Science* (1992) 60–69.

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

[7] G. N. Frederickson. Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. *32nd IEEE Symp. Foundations of Computer Science* (1991) 632–641.

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

[9] J. Mulvey. Pivot strategies for primal-simplex network codes. *J. Assoc. Comput. Mach.* 25 (1978) 266–270.

[10] S. A. Plotkin and É. Tardos. Improved dual network simplex. *1st ACM-SIAM Symp. Discrete Algorithms* (1990) 367–376.

[11] M. Rauch. Fully dynamic biconnectivity in graphs. *33rd IEEE Symp. Foundations of Computer Science* (1992) 50–59.

[12] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.* 24 (1983) 362–381.

[13] P. M. Vaidya. Speeding up linear programming using fast matrix multiplication. *30th IEEE Symp. Foundations of Computer Science* (1989) 332–337.