Dynamic trees in exterior-point Simplex-type algorithms for network flow problems

George Geranis [a,1] Angelo Sifaleras [b,2]

[a] *Department of Applied Informatics, University of Macedonia, 156 Egnatia Str., 54006 Thessaloniki, Greece*

[b] *Department of Technology Management, University of Macedonia, Loggou-Tourpali, 59200 Naoussa, Greece*

Abstract

Recently, a new Dual Network Exterior-Point Simplex Algorithm (DNEPSA) for the Minimum Cost Network Flow Problem (MCNFP) has been developed. In extensive computational studies, DNEPSA performed better than the classical Dual Network Simplex Algorithm (DNSA). In this paper, we present for the first time how to utilize the dynamic trees data structure in the DNEPSA algorithm, in order to achieve an improvement of the amortized complexity per pivot. Our work constitutes a first step towards the development of an efficient implementation of DNEPSA.

Keywords: Network Optimization, Computational Complexity, Data Structures.

1 Introduction

Network optimization [1] is a core area of combinatorial optimization, consisting of optimization problems that can be modeled using networks. Important theoretical improvements in network optimization algorithms have been

[1] Email: `geranis@uom.gr`
[2] Email: `sifalera@uom.gr`

achieved using well-known efficient data structures like dynamic trees [9], or Fibonacci heaps [2]. The MCNFP [8] is the problem of finding a minimum cost flow of product units, through a number of source nodes, sinks and transshipment nodes.

Let $G = (N, A)$ be a directed network, where two finite sets N and A consist of n nodes and m directed arcs, respectively. For each node $i \in N$, there is an associated numeric value b_i. A node i is a source node if it is $b_i > 0$, a sink node if it is $b_i < 0$ and a transshipment node otherwise. The problem will be called balanced, if the total supply is equal to the total demand, i.e., $\sum_{i\in N} b_i = 0$. For each arc $(i, j) \in A$, there is an associated flow x_{ij} and an associated cost c_{ij}, showing the amount of product units transferred from node i to node j and the flow cost per product unit, respectively. An optimal solution of the problem is a flow consisting of arc flows $x_{ij} \geq 0$ that minimize the total cost $\sum_{(i,j)\in A} c_{ij}x_{ij}$, subject to the flow conservation equations $\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = b_i$, $\forall i \in N$. In this paper, DNEPSA algorithm with dynamic trees, is applied to uncapacitated MCNFPs (i.e., $0 \leq x_{ij} \leq +\infty$). The dual problem can be formulated using a set of dual variables w_i and a set of reduced cost variables s_{ij}, one for each node $i \in N$ and for each arc $(i, j) \in A$, respectively.

A number of different algorithmic approaches have been developed for the MCNFP, including the well-known Primal and Dual Network Simplex Methods. A primal exterior-point Simplex-type algorithm for the MCNFP was presented in [7] extending other approaches, e.g., to the assignment problem [6]. The DNEPSA [3] for the MCNFP is an exterior Simplex-type algorithm that starts from an initial dual feasible tree-solution and, after a number of iterations, it reaches an optimal solution by producing a sequence of tree-solutions that can be both dual and primal infeasible. This paper describes the representation of the tree-solutions produced by DNEPSA by using dynamic trees. Section 2 presents a short description of DNEPSA. The tree operations that improve the algorithm's theoretical performance, are described in detail for each step of the algorithm in Section 3. Finally, Section 4 presents conclusions and ongoing research efforts.

2 Description of DNEPSA

Computational results [3] on randomly generated network problems, have already shown the superiority of DNEPSA to DNSA (both on the number of the iterations and on the time needed in order to find an optimal solution), even using the Augmented Thread Index method (ATI method) [4] rather than the more efficient data structure of dynamic trees. However, it is well known that the use of special data structures for storing and updating the necessary variables can improve the algorithm's amortized complexity and also its computational performance.

Once a starting dual feasible tree solution has been computed, i.e., the flows x_{ij}, the node potentials (dual variables) w_i, and the reduced costs s_{ij}, as described in [5], then the steps of DNEPSA are as follows:

Step 0 (Initialization). Start with a partition of the basic arcs into two subsets as follows $I_- = \{(i,j) \in T : x_{ij} < 0\}$ and $I_+ = \{(i,j) \in T : x_{ij} \geq 0\}$. Compute a direction flow vector d, towards the feasible region of the dual problem, using the following relations: $d(I_-) = 1, d(I_+) = 0, d_{ij} = \sum_{(u,v)\in I_-} h_{uv}, \forall (i,j) \notin T$.

Step 1 (Termination test). If $I_- = \emptyset$, this means that the current tree-solution is optimal. Otherwise, create a set $J_- = \{(i,j) \notin T : s_{ij} > 0 \wedge d_{ij} < 0\}$. If $(J_- = \emptyset) \wedge (I_- \neq \emptyset)$, then the problem is infeasible.

Step 2 (Choice of entering arc). Choose the entering arc (g,h) using the following minimum ratio test: $\alpha = \frac{s_{gh}}{-d_{gh}} = \min\left\{\frac{s_{ij}}{-d_{ij}} : (i,j) \in J_-\right\}$. An entering arc can be found in $O(m)$ time.

Step 3 (Choice of leaving arc). Find the minimum ratios: $\theta_1 = -x_{k_1 l_1} = \min\{-x_{ij} : (i,j) \in I_- \wedge (i,j) \uparrow\uparrow (g,h)\}$ and $\theta_2 = x_{k_2 l_2} = \min\{x_{ij} : (i,j) \in I_+ \wedge (i,j) \uparrow\downarrow (g,h)$. Choose the leaving arc $(g,h) = (k_1,l_1)$ or (k_2,l_2), in case that $\theta_1 \leq \theta_2$ or not respectively. A leaving arc can be found in $O(n)$ time

Step 4 (Pivoting). If $\theta_1 \leq \theta_2$, the pivot will be called "Type A iteration". In this case, set $I_- = I_- \setminus (k,l)$ and $I_+ = I_+ \cup (g,h)$. An arc of negative flow $x_{kl} = -\theta_1$ is leaving the basic tree-solution T and an arc of positive flow $x_{gh} = \theta_1$ is entering T. Otherwise, if it is $\theta_1 > \theta_2$, then $(k,l) = (k_2,l_2)$ is the leaving arc and we have a *type B iteration*. In this case, set $I_+ = I_+ \cup (g,h) \setminus (k,l)$. An arc of positive flow $x_{kl} = \theta_2$ is leaving T and an arc with positive flow $x_{gh} = \theta_2$ is entering T. Let T^* be the sub-tree that is being cut off from the basic tree solution, if we first remove the leaving arc. Values x_{ij}, s_{ij}, d_{ij} or sets I_-, and I_+ can be efficiently updated from iteration to iteration as described in [3], rather than computed from scratch in each iteration. The update of the basic tree solution using ATI method requires $O(n)$ time [1]

3 Using Dynamic Trees in DNEPSA

3.1 Description of DNEPSA using dynamic trees

Dynamic trees [9], are data structures that maintain a set of vertex disjoint rooted trees and allow us to easily change their structure by using mainly three kinds of operations:

- $link(u, v)$: links together two dynamic trees, where u is the root of the first tree and v a vertex of the second tree, by adding the edge (u, v).
- $cut(v)$: divides a dynamic tree into two subtrees by deleting the edge that connects v to its parent.
- $evert(v)$: makes v the root of the tree by turning the tree "inside out".

Beyond these three basic operations, there are other useful operations that can be implemented, such as:

- $parent(v)$: returns the parent of a vertex.
- $root(v)$: returns the root of the dynamic tree containing v.
- $cost(v)$: returns the cost of the edge $(v, parent(v))$.
- $mincost(v)$: finds the edge of minimum cost on the path joining v to $root(v)$.
- $update(v, x)$: adds value x to all edges on the path from v to $root(v)$.

A dynamic tree can be partitioned into a collection of vertex-disjoint dynamic paths by dividing its edges into two types: solid and dashed edges. At most one solid path can enter any vertex of the tree. Thus, a dynamic path is determined by a sequence of solid edges and a dynamic tree is partitioned into a number of dynamic paths connected together by dashed edges. Each tree-solution is a collection of dynamic paths connected to each other by dashed lines where, each dynamic path is implemented by a biased binary tree. A basic tree solution in DNEPSA can be represented by a directed dynamic tree consisting of a set of vertex-disjoint dynamic paths. Figure 1 shows a possible initial dual feasible tree-solution used by DNEPSA in order to solve a MCNFP problem. The value next to a vertex v represents its supply or demand and the value next to an arc (i, j) represent the flow x_{ij} for that arc. Such a tree-solution can be built by a special algorithm, like the algorithm described in [5], that starts from a collection of $|N|$ single-vertex dynamic trees and uses link operations to construct the final tree. The type of the arcs of the tree (solid or dashed) depends on the order that linking operations take place.

The dynamic tree in Figure 1 consists of three dynamic paths: (8, 2, 4, 6), (7, 3), and (5,1). The head of a path is its bottommost vertex and the tail

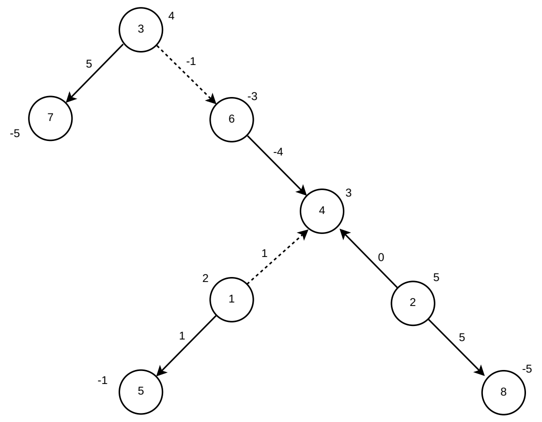

Fig. 1. Representation of the initial dual feasible tree as a dynamic tree.

of the path is its topmost vertex. A set of primitive dynamic path operations have to be implemented:

- $path(v)$: returns the path containing vertex v.
- $head(p)$ and $tail(p)$: return the first and the last node of path p.
- $before(v)$ and $after(v)$: return the previous and next node of v.
- $pcost(v)$: returns the cost of edge $(v, after(v))$.
- $pmincost(p)$: finds a node v on path p so that the edge $(v, after(v))$ has the minimum cost.
- $pupdate(p, x)$: adds x to the cost of every edge of path p.
- $reverse(p)$: reverses the direction of the path.
- $concatenate(p, q, x)$: concatenates two paths p and q by adding a new edge of cost x.
- $split(v)$: deletes edges $(before(v), v)$ and $(v, after(v))$ producing two subpaths.

In addition, two composite path operations are needed:

- $splice(p)$: transforms the dashed edge leaving $tail(p)$ into a solid edge; thus the path is being extended this way.
- $expose(v)$: transforms every dashed edge from v to $root(v)$ into a solid edge.

These composite operations are implemented by using the primitive operations described before. The dynamic tree operations we need, are implemented by using some of the above primitive and composite path operations. Dynamic paths can be implemented as biased binary trees. Figure 2 shows the representation of the dynamic tree in Figure 1 as three different binary trees, one for each dynamic path. The external nodes of a dynamic path in left-to-right order correspond to the vertices of the path from head to tail. An internal node w of the tree corresponds to the edge of the path that connects nodes u and v, where u and v are the previous and next node of the tree when it is traversed in symmetric order (inorder traversal). For all vertices of the dynamic tree with an outgoing dashed line, there is a field named *dparent* that connects a vertex to its parent in the original dynamic tree and a field named *dcost* that stores the corresponding cost.

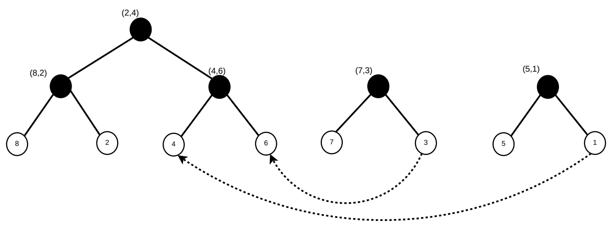

Fig. 2. Representation of dynamic paths (8, 2, 4, 6), (7, 3) and (5,1) as binary trees.

For each node of the binary trees used to represent dynamic paths, a number of fields can be used in the data structure to store the required information:

- *external* field: indicates if the node is internal or external (values 0 or 1).
- *direction* field: stores the direction of the arc that corresponds to an internal node.
- *reversed* field: shows if an arc was reversed.
- *bparent* field: points to the parent of a node (it is null for the root).
- *bhead* and *btail* fields: point to the head and tail of the path.
- *bleft* and *bright* fields: point to the left and right child of an internal node of the tree.

When used in DNEPSA, we need more fields in order to store information about the flow of the arc that corresponds to an internal node and a flag indicating whether it belongs to I_- or I_+. We need also to store supply values b_i and dual variable values w_i in every external node of a binary tree.

Sleator and Tarjan presented an implementation of dynamic trees, based on self-adjusting binary trees (splay trees), that requires $O(\log n)$ amortized time per one dynamic-tree operation, by representing solid paths as locally biased binary trees. Thus, by using dynamic trees and utilizing these results it is possible to improve the amortized time complexity of DNEPSA per iteration for updating the basic tree solution, as it is described in subsection 3.2.

3.2 Analysis of DNEPSA using dynamic trees

In step 0, the partition of subsets I_- and I_+ requires $O(n)$ time, and the computation of the flow vector d requires $O(m)$ time. In step 1, if $I_- \neq \emptyset$ then we need to compute the J_- set and the reduced cost variables s_{ij}, that requires $O(m)$ time. In step 2, finding the entering arc (g, h) using a minimum ratio α, also require $O(m)$ time.

In step 3, DNEPSA has to select the leaving arc. The flows in cycle C existing in $T \cup (g, h)$ have to be checked so that θ_1 and θ_2 are computed. By using dynamic trees, in order to compute θ_1 and θ_2, we can first make h to be the root of the dynamic tree (by using operation *evert(h)*), then *expose(g)*, and then apply a "modified" operation *mincost(g)*, that checks also if a basic arc belongs in I_- or I_+ and compares the orientation of these arcs.

In step 4, after finding the entering and the leaving arc, the new tree-solution $T \setminus (k, l) \cup (g, h)$ has to be built. This can be easily done by using the *cut* and *link* dynamic tree operations. The entering arc (g, h) is added into I_+ by setting the proper bit in the node's structure and its flow x_{gh} becomes equal to θ_1 or θ_2, depending on the type of iteration (*type A* or *type B*).

The flows x_{ij} for the basic arcs $(i, j) \notin C$ do not change. On the other hand, for a basic arc $(i, j) \in C$ the flow x_{ij} changes and its new value $x_{ij}^{(t+1)}$ is either $x_{ij}^{(t)} - x_{kl}^{(t)}$ or $x_{ij}^{(t)} + x_{kl}^{(t)}$ depending on the orientation of (i, j) compared to the orientation of (g, h). These operations can be also done by using operations *evert(h)*, *expose(g)*, and finally apply a "modified" operation *update(g,* $x_{kl}^{(t)}$*)*. Dynamic tree operations, in the same way as previously described, can help in determining the orientation of an arc $(i, j) \in C$ in $O(\log n)$ time.

The dual variables w_i corresponding to the nodes belonging in T^* can be also updated in a similar way; the value of the remaining dual variables do not change. Again the time complexity for these "modified" tree operations is $O(\log n)$. Thus, DNEPSA overall has an $O(\log n)$ amortized time complexity bound per iteration for updating the basic tree solution, by using dynamic trees.

4 Conclusions and Future Work

By using dynamic trees in DNEPSA as described in the previous section, the amortized time complexity per pivot for updating the basic tree solution is $O(\log n)$, giving an improvement over the previous algorithm implementation using the ATI method. However, the overall running time is not improved, since it is dominated by the selection of the entering arc.

Apart from the theoretical findings, regarding the amortized time complexity bound per iteration, experimental results demonstrating the efficiency of the proposed approach are also of equal importance. Therefore, our ongoing research efforts focus on computational testing of the DNEPSA performance using dynamic trees in order to provide empirical support for the reduction in time per iteration.

Acknowledgement

This research has been partly funded by the Research Committee of the University of Macedonia, Economic and Social Sciences, Greece, under grant 80749 for the advance of Basic Research.

References

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

[2] Fredman, M. L. and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, Journal of the ACM **34** (1987), pp. 596–615.

[3] Geranis, G., K. Paparrizos and A. Sifaleras, *On a dual network exterior point simplex type algorithm and its computational behavior*, RAIRO - Operations Research **46** (2012), pp. 211–234.

[4] Glover, F., D. Karney and D. Klingman, *The augmented predecessor index method for locating stepping stone paths and assigning dual prices in distribution problems*, Transportation Science **6** (1972), pp. 171–180.

[5] Hultz, J., D. Klingman and R. Russell, *An advanced dual basic feasible solution for a class of capacitated generalized networks*, Operations research **24** (1976), pp. 301–313.

[6] Papamanthou, C., K. Paparrizos, N. Samaras and A. Sifaleras, *On the initialization methods of an exterior point algorithm for the assignment problem*, International Journal of Computer Mathematics **87** (2010), pp. 1831–1846.

[7] Paparrizos, K., N. Samaras and A. Sifaleras, *A new exterior simplex type algorithm for the minimum cost network flow problem*, Computers & Operations Research **36** (2009), pp. 1176–1190.

[8] Sifaleras, A., *Minimum cost network flows: Problems, algorithms, and software*, Yugoslav Journal of Operations Research **23** (2013), pp. 3–17.

[9] Sleator, D. D. and R. E. Tarjan, *A data structure for dynamic trees*, Journal of Computer and System Sciences **26** (1983), pp. 362–391.