Information Processing Letters, Vol. 97(3), pp. 109-114, 2006

A Fixed-Parameter Tractability Result for Multicommodity Demand Flow in Trees [1]

Jiong Guo and Rolf Niedermeier

Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany.
{guo,niedermr}@minet.uni-jena.de.

Abstract

We study an NP-hard (and MaxSNP-hard) problem in trees—Multicommodity Demand Flow—dealing with demand flows between pairs of nodes and trying to maximize the value of the routed flows. This problem has been intensively studied for trees as well as for general graphs mainly from the viewpoint of polynomial-time approximation algorithms. By way of contrast, we provide an exact dynamic programming algorithm for this problem that works well whenever some natural problem parameter is small, a reasonable assumption in several applications. More specifically, we prove fixed-parameter tractability with respect to the maximum number of the input flows at any tree node.

Keywords. Combinatorial problems, graph algorithms, NP-hard problems, exact algorithms, fixed-parameter tractability.

1 Introduction

As a rule, most hard problems on graphs turn easy when restricted to trees. There are, however, notable exceptions of important graph problems that remain hard even in trees. A well-known example is the Bandwidth Minimization problem restricted to trees of maximum degree three where it still remains NP-complete [10]. In this paper, we deal with another well-studied graph problem that remains NP-complete when restricted to trees. The problem is Multicommodity Demand Flow in Trees (MDFT), where the tree edges have limited capacities, each demand (i.e., flow) between two nodes

[1] Supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4. Parts of this work have been done while the authors were at the Universität Tübingen.

has a certain value and each successfully routed demand brings a certain profit; the goal is to maximize the profit by routing as many and as good flows as possible. We refer to Sect. 2 for more details and formal definitions. It follows from the work of Garg, Vazirani, and Yannakakis [8] that this problem is NP-complete and MaxSNP-hard. Moreover, constant-factor polynomial-time approximation algorithms are known [4,8]. By way of contrast, we investigate the exact solvability of this problem by fixed-parameter algorithms.

Exact algorithms for NP-hard problems have become a flourishing field of research [6,9,12,13]. In particular, fixed-parameter algorithms are now often seen as valuable alternatives to approximation algorithms. The basic idea is to try to derive exact algorithms with a combinatorial explosion (i.e., exponential running time factor) that can be confined to some (hopefully small) problem parameters. Formally, a parameterized problem is *fixed-parameter tractable* if it has a solving algorithm that runs in $O(f(k) \cdot n^c)$ time, where f is an arbitrary computable function only depending on k, k is the input parameter, n is the problem size, and c is a constant [6,12] (see [5,7,11] for recent surveys). Meanwhile, many successful examples of fixed-parameter algorithms for hard problems exist. Here, we contribute one further example of this sort, complementing previous work on approximation and fixed-parameter algorithms. More precisely, we show that MDFT is fixed-parameter tractable with respect to the parameter "maximum number of the input flows running through or to any node of the input tree network." Interestingly, as further discussed in the concluding section, there is no hope for fixed-parameter tractability for MDFT when replacing in the parameter the term "node" by "edge". Our new fixed-parameter algorithm for MDFT is conceptually simple enough to allow easy implementation and, also because of that, may appear as profitable alternative to existing approximation algorithms.

2 Problem Definition and Previous Results

The Multicommodity Demand Flow in Trees (MDFT) problem is defined as follows.

Input: A tree network $T = (V, E)$, where each edge e is associated with an integer $c(e) > 0$ as its capacity, and a collection F of flows which is encoded as a list of pairs of nodes of T,

$$F = \{f_i \mid f_i := (u_i, v_i), u_i \in V, v_i \in V, u_i \neq v_i, 1 \leq i \leq m\}.$$

Each flow $f \in F$ has associated an integer demand value $d(f) > 0$ and a real valued profit $p(f) > 0$.
Task: Find a routable subset $F' \subseteq F$ which maximizes $p(F') := \sum_{f \in F'} p(f)$.

A subset $F' \subseteq F$ is *routable* (in T) if the flows in F' can be simultaneously routed without violating any edge capacity of the tree. [2]

The tree T is termed the *supply tree*. Throughout this paper let $n := |V|$ and $m := |F|$. The nodes u, v are called the two *endpoints* of the flow $f = (u, v)$. Note that, for a tree, the path between two distinct nodes is uniquely determined and can be found in linear time. Thus, we can assume that each flow $f = (u, v) \in F$ is given as a path between u and v. We use F_v to denote the set of demand flows passing through node v. A demand flow *passes through* node v if it has v as one of its endpoints or v lies on the flow's path.

It follows from the results of Garg, Vazirani, and Yannakakis that MDFT is NP-complete and MaxSNP-hard even in the case of unit demands and unit profits [8]. For this special case of unit demands and unit profits, they gave a factor-2 polynomial-time approximation algorithm. Recently, Chekuri, Mydlarz, and Shepherd [4] showed that in the special case of MDFT with unit demands but arbitrary profits the natural linear programming relaxation yields a factor-4 approximation. They further showed that, under the assumption that the maximum flow demand is at most the minimum edge capacity, in the general case with arbitrary demands and arbitrary profits the natural linear programming relaxation of MDFT has an integrality gap of at most 48, improving previous work of Chakrabarti et al. [3] which dealt with path instead of tree networks. We refer to Chekuri et al. [4] and Garg et al. [8] with respect to further relevance of studying MDFT. Both papers, however, concentrate on the polynomial-time approximability of MDFT and its still NP-complete special cases. Concerning exact algorithms, we are only aware of the recent work of Anand et al. [1] where (a special case of) MDFT is referred to as "call admission control problem." For the special case of MDFT restricted to instances with unit demands and unit profits, they presented a fixed-parameter algorithm with running time $O(2^d \cdot d! \cdot |I|^{O(1)})$, where $|I|$ denotes the size of the input instance and d denotes the number of flows to be rejected in order to "enable" all remaining flows. Hence, their version of MDFT is fixed-parameter tractable with respect to parameter d defined in this way.

By way of contrast, for the general MDFT problem, we subsequently show that it is fixed-parameter tractable with respect to the new parameter $k :=$ "maximum number of the input flows passing through any node of the tree network," i.e., $k := \max_{v \in V} |F_v|$. The running time is $O(2^k \cdot mn)$. The corresponding fixed-parameter algorithm is superior to approximative solutions whenever k is of limited size, a realistic assumption for several applications.

[2] That is, for any edge e the sum of the demand values of the flows routed through e does not exceed $c(e)$.

$a := a_1 a_2 \dots a_{k_v}$	D_e
$00 \dots 00$	
$00 \dots 01$	
$\vdots$	
$11 \dots 11$	

Fig. 1. Table D_e for edge $e = (u, v)$ with $F_v := \{f_1^v, f_2^v, \dots, f_{k_v}^v\}$ with $k_v \leq k$

3 The Algorithm

The fixed-parameter algorithm is based on dynamic programming. We begin with some agreements that simplify the presentation and analysis of the algorithm. Then, we describe the algorithm in detail and after that we prove its correctness and analyze its time complexity. Finally, we briefly discuss a very special case of MDFT on paths which can be easily solved in linear time.

3.1 Agreements and Basic Tools

We assume that we deal with arbitrarily rooted trees. Thus, an edge $e = (u, v)$ reflects that u is the parent node of v. In particular, using the rooted tree structure, we will solve MDFT in a bottom-up fashion by dynamic programming from the leaves to the root. In this context, we use $T[v]$ to denote the subtree of the input tree rooted at node v.

The core idea of the dynamic programming is based on the following definition of tables which are used throughout the algorithm. For each edge $e = (u, v)$ with $F_v := \{f_1^v, f_2^v, \dots, f_{k_v}^v\} \subseteq F$ we construct a table D_e as illustrated in Fig. 1. Table D_e has 2^{k_v} rows which correspond to all possible k_v-vectors a over $\{0, 1\}$, i.e., binary vectors having k_v components. Each of these vectors represents a *route schedule* for the flows in F_v. The ith component a_i of a corresponds to flow f_i^v, and $a_i = 0$ means that we do not route flow f_i^v and $a_i = 1$ means that we do route f_i^v. Furthermore, to refer to the set of routed flows, we define $r(a) := \{i \mid a_i = 1, 1 \leq i \leq k_v\}$. Table entry $D_e(a)$ stores the maximum profit which we can achieve according to the route schedule encoded by a with the flows which have at least one of their endpoints in $T[v]$.

3.2 Dynamic Programming Algorithm

The algorithm works bottom-up from the leaves to the root. Having computed all tables D_e for the edges e connected to the root node, MDFT will be solved easily as pointed out later on. The algorithm starts with "leaf edges" which are initialized as follows. For an edge $e = (u, v)$ connecting a leaf v with its parent node u, the table entries for the at most 2^k rows a can be easily computed as

$$D_e(a) := \begin{cases} 0, \text{ if } c(e) < \sum_{i \in r(a)} d(f_i^v); \\ \sum_{i \in r(a)} p(f_i^v), \text{ otherwise.} \end{cases}$$

Then, the main algorithm consists of distinguishing between three cases.

Case 1. Consider an edge $e = (u, v)$ connecting two non-leaf nodes u and v where v has only one child w connected to v by edge $e' = (v, w)$.

The sets of flows passing through nodes u, v, and w are denoted by $F_u := \{f_1^u, \ldots, f_{k_u}^u\}$, $F_v := \{f_1^v, \ldots, f_{k_v}^v\}$, and $F_w := \{f_1^w, \ldots, f_{k_w}^w\}$. Moreover, we use F_e and $F_{e'}$ to denote the sets of flows passing through e and e' and we have $F_e = F_u \cap F_v$ and $F_{e'} = F_v \cap F_w$. First, if $F_e \cap F_{e'} = \emptyset$, then the given instance can be divided into two smaller instances, one consisting of subtree $T[v]$ and the flows inside it and the other consisting of the original tree without the nodes below v and the flows therein. The optimal solution for the original instance then is the sum of the optimal solutions of the two smaller instances. An optimal solution of the first smaller instance is already computed and it is obtained from a maximum entry of table $D_{e'}$. In order to compute an optimal solution of the second smaller instance, we can treat v as a leaf and proceed as for leaf edges as described above.

Second, if $F_e \cap F_{e'} \neq \emptyset$, then there are some flows passing through both e and e'. Recall that entry $D_e(a)$ for a k_v-vector a shall store the maximum profit with respect to the route schedule encoded by a that can be achieved by the flows with at least one of their endpoints in $T[v]$. We partition F_v into two sets, $F_v \cap F_w$ and $F_v \setminus F_w$. The value of $D_e(a)$ is thus the sum of the maximum of the entries of $D_{e'}$ which have the same route schedule for the flows in $F_v \cap F_w$ as encoded in a, and the profit achieved by the flows in $F_v \setminus F_w$ obeying the route schedule encoded by a. Let $B^v := \{i \mid f_i^v \in (F_v \cap F_w)\}$, $B^w := \{i \mid f_i^w \in (F_v \cap F_w)\}$, and $j := |B^v| = |B^w|$.[3] To easier obtain the maximum of the entries of $D_{e'}$ which have the same route schedule for the

[3] Clearly, B^v and B^w refer to the same sets of demand flows. Note, however, that they may differ due to different "naming" of the same flow in the two tables D_e and $D_{e'}$.

flows in $F_v \cap F_w$, we condense table $D_{e'}$ with respect to B^w. The *condensation of $D_{e'}$ with respect to B^w* is to keep only the components of the k_w-vector a' of $D_{e'}$ which correspond to the demand flows in $F_v \cap F_w$. More precisely, for a route schedule $\overline{a'}$ condensed with respect to B^w, we obtain

$$D_{e'}(\overline{a'}) := \max\{D_{e'}(a') \mid \overline{a'} = \pi_{(B^w)}(a')\}.$$

Herein, $\pi_{(B^w)}(a')$ returns the projection of a' onto the j components of a' that correspond to B_w. Then, using $A := \{i \mid f_i^v \in F_e\}$ to refer to the set of the flows in F_v passing through edge e, the entries of D_e are computed as follows.

$$D_e(a) := \begin{cases} 0, \text{ if } c(e) < \sum_{i \in A} d(f_i^v); \\ D_{e'}(\pi_{(B^v)}(a)) + \sum_{i \in (r(a) \setminus B^v)} p(f_i^v), \text{ otherwise.} \end{cases}$$

Obeying the route schedule encoded by the k_v-vector a, the terms $D_{e'}(\pi_{B^v}(a))$ and $\sum_{i \in (r(a) \setminus B^v)} p(f_i^v)$ denote the profits achieved by the flows in $F_v \cap F_w$ and in $F_v \setminus F_w$, respectively.

Case 2. Consider an edge $e = (u, v)$ connecting two non-leaf nodes u and v where v has two children w_1 and w_2 connected to v by edges $e' = (v, w_1)$ and $e'' = (v, w_2)$.

We use $F_u := \{f_1^u, \ldots, f_{k_u}^u\}$, $F_v := \{f_1^v, \ldots, f_{k_v}^v\}$, $F_{w_2} := \{f_1^{w_1}, \ldots, f_{k_{w_1}}^{w_1}\}$, and $F_{w_2} := \{f_1^{w_2}, \ldots, f_{k_{w_2}}^{w_2}\}$ to denote the sets of flows passing through nodes u, v, w_1, and w_2. As in Case 1, $F_e = F_u \cap F_v$, $F_{e'} = F_v \cap F_{w_1}$, and $F_{e''} = F_v \cap F_{w_2}$. With the same argument as in Case 1, if one of $F_e \cap F_{e'}$ and $F_e \cap F_{e''}$ is empty, we can divide the given instance into two smaller instances and solve them separately. Thus, we assume that they are not empty. Similar to Case 1, we "partition" F_v into $F_v \cap F_{w_1}$, $F_v \cap F_{w_2}$, and $(F_v \setminus F_{w_1}) \setminus F_{w_2}$. For a k_v-vector a, one might simply set $D_e(a)$ equal to the sum of the maximum of the entries of $D_{e'}$ which have the same route schedule as encoded in a for the flows in $F_v \cap F_{w_1}$, the maximum of the entries of $D_{e''}$ which have the same route schedule as encoded in a for the flows in $F_v \cap F_{w_2}$, and the profit achieved by the flows in $(F_v \setminus F_{w_1}) \setminus F_{w_2}$ obeying the route schedule encoded by a. However, if $(F_v \cap F_{w_1}) \cap (F_v \cap F_{w_2}) \neq \emptyset$, that is, due to the tree structure, $F_{w_1} \cap F_{w_2} \neq \emptyset$, then the edges e' and e'' have some common flows. Then, for each flow between $T[w_1]$ and $T[w_2]$ scheduled to be routed in both subtrees, we have to once subtract its profit from the sum to avoid double counting. Let

$$B_1^v := \{i \mid f_i^v \in (F_v \cap F_{w_1})\}; \qquad B_1^{w_1} := \{i \mid f_i^{w_1} \in (F_v \cap F_{w_1})\};$$
$$B_2^v := \{i \mid f_i^v \in (F_v \cap F_{w_2})\}; \qquad B_2^{w_2} := \{i \mid f_i^{w_2} \in (F_v \cap F_{w_2})\};$$
$$B_3^{w_1} := \{i \mid f_i^{w_1} \in (F_{w_1} \cap F_{w_2})\}; \quad B_3^{w_2} := \{i \mid f_i^{w_2} \in (F_{w_1} \cap F_{w_2})\}.$$

Note that $|B_1^v| = |B_1^{w_1}|$, $|B_2^v| = |B_2^{w_2}|$, $|B_3^{w_1}| = |B_3^{w_2}|$, $B_3^{w_1} \subseteq B_1^{w_1}$, and $B_3^{w_2} \subseteq$

$B_2^{w_2}$. As in Case 1, we condense $D_{e'}$ and $D_{e''}$. More specifically, we condense $D_{e'}$ with respect to $B_1^{w_1}$ and $D_{e''}$ with respect to $B_2^{w_2}$:

$$D_{e'}(\overline{a'}) := \max\{D_{e'}(a') \mid \overline{a'} = \pi_{(B_1^{w_1})}(a')\};$$
$$D_{e''}(\overline{a''}) := \max\{D_{e''}(a'') \mid \overline{a''} = \pi_{(B_2^{w_2})}(a'')\}.$$

Then, using $A := \{i \mid f_i^v \in F_e\}$, the entries of D_e are computed as follows.

$$D_e(a) := \begin{cases} 0, \text{ if } c(e) < \sum_{i \in A} d(f_i^v); \\ \alpha + \beta, \text{ otherwise.} \end{cases}$$

Herein,

$$\alpha := D_{e'}(\pi_{(B_1^v)}(a)) + D_{e''}(\pi_{(B_2^v)}(a)) - \gamma,$$
$$\beta := \sum_{i \in (r(a) \setminus (B_1^v \cup B_2^v))} p(f_i^v),$$
$$\gamma := \sum_{i \in B_3^{w_1},\, \pi_{(\{i\})}(a)=1} p(f_i^{w_1}).$$

In order to avoid double counting of common flows of edges e' and e'', γ has been subtracted in the above determination of D_e.

Case 3. Consider an edge $e = (u, v)$ connecting two non-leaf nodes u and v where v has $l > 2$ children $w_1, w_2, \ldots, w_l$.

We add $l-2$ new nodes $v_1, v_2, \ldots, v_{l-2}$ to T and we transform T into a binary tree. Each of these new nodes has exactly two children. Node v_1 is the parent node of w_1 and w_2, v_2 is the parent node of v_1 and w_3, and so on. Thus, v becomes the parent node of v_{l-2} and w_l. The edge between w_i and its new parent node is assigned the same capacity as the edge between w_i and v in the original tree. The edges $(v, v_{l-2}), (v_{l-2}, v_{l-1}), \ldots, (v_2, v_1)$ between the new nodes obtain unbounded capacity. The flows have the same endpoints as in the original instance. It is easy to see that the solutions for the new and the old tree are the same, and thus Case 1 and Case 2 suffice for handling the new binary tree. This concludes the description of the dynamic programming algorithm.

3.3 Main Result

The above described dynamic programming algorithm leads to the following.

Theorem 1 MDFT *can be solved in* $O(2^k \cdot mn)$ *time, where* k *denotes the maximum number of demand flows passing through any node of the given supply tree, i.e.,* $k := \max_{v \in V} |F_v|$.

PROOF. The correctness of the algorithm basically follows directly from its description. To this end, however, note that when the D-tables of all edges of the supply tree are computed, we may easily determine the final optimum by comparing the tables of the without loss of generality at most two edges leading to the root. Moreover, by means of a top-down traversal from the root to the leaves we can easily determine the actual subset of routable flows that led to the overall maximum profit. We omit the straightforward details here.

Concerning the algorithm's running time, observe that table D_e for edge $e = (u, v)$ has at most $O(2^k)$ entries. For a node v with more than two children, as described in Sect. 3.2, we add some new nodes to construct a binary tree. The resulting tree at most doubles in size. Moreover, since $F_{v'} \subseteq F_v$ for each of the new nodes v', the D-tables of the new edges have at most $O(2^k)$ entries. Assuming that all basic set operations such as union, set minus, etc. between two sets having at most m elements can be done in $O(m)$ time, the computation of a new table from its at most two child tables takes $O(2^k \cdot m)$ time. Altogether, the algorithm then takes $O(2^k \cdot mn)$ time. □

3.4 A Trivial Version of MDFT Restricted to Paths

Chekuri et al. [4] mention that MDFT restricted to paths and unit flow demands can be solved in polynomial time by linear programming or combinatorially by a minimum cost circulation problem [2]. We describe a very simple linear time algorithm that solves this problem when further restricted to unit profits.

The algorithm starts at the left end of the supply path. While not reaching the right end of the path, it does the following.

(1) Check whether the capacity of the current edge $e = (u, v)$ suffices, where u is the left endpoint of e and v is the right endpoint.
(2) If not, remove "longest" demand flows which start in u until the capacity of e suffices.
(3) Replace each of the flows $f = (u, z)$ with $z \neq v$ by a flow between v and z.

The solution is simply given by the set of non-removed demand flows.

Proposition 2 *MDFT restricted to paths as supply tree, unit flow demands, and unit profits can be solved in $O(m+n)$ time.*

PROOF. The above algorithm is obviously correct. After simple preprocessing, we know the length of each demand flow. Then, we can proceed from left to right in $O(n)$ main steps. All steps together need to remove at most $O(m)$ demand flows, yielding $O(m+n)$ running time. □

4 Conclusion

Employing dynamic programming, we obtained a fixed-parameter tractability result for Multicommodity Demand Flow in Trees. Our result complements previous work mainly dealing with the polynomial-time approximability of this and related problems. We claim that our exact algorithm is conceptually simple enough to be valuable in practice. Clearly, the immediate challenge is to significantly improve the exponential time bound of our algorithm.

Note that the parameter in our fixed-parameter tractability result relates to the maximum number of flows passing through any *node* of the input tree. The natural question arises what happens when we replace "node" by "*edge*". Somewhat surprisingly, a simple adaption of an NP-completeness proof of Garg et al. [8, Theorem 4.2] shows that MDFT is NP-complete even when there are at most six demand flows passing through an edge. Hence there is no hope for fixed-parameter tractability with respect to the parameter "maximum number of demand flows passing through any edge of the tree network". Finally, as pointed out by an anonymous referee, it seems worth studying the complexity of MDFT when parameterized by the number of the demand flows through any node in a *solution* instead of the number of demand flows through any node in the *input* (as we did here). We leave it as an open question to investigate the parameterized complexity for this modified parameterization.

Acknowledgment. We are grateful to an anonymous referee of *Information Processing Letters* for pointing out an ambiguity in our previous definition of MDFT.

References

[1] R. S. Anand, T. Erlebach, A. Hall, and S. Stefanakos. Call control with k rejections. *Journal of Computer and System Sciences*, 67(4): 707–722, 2003.

[2] G. Călinescu, A. Chakrabarti, H. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *Proc. of 9th IPCO 2002*, volume 2337 of LNCS, pages 401–414, Springer, 2002.

[3] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. In *Proc. of 5th APPROX 2002*, volume 2462 of LNCS, pages 51–66, Springer, 2002.

[4] C. Chekuri, M. Mydlarz, and F. B. Shepherd. Multicommodity demand flow in a tree (extended abstract). In *Proc. of 30th ICALP 2003*, volume 2719 of LNCS, pages 410–425, Springer, 2003.

[5] R. G. Downey. Parameterized complexity for the skeptic. In *Proc. of 18th IEEE Conference on Computational Complexity*, pages 147–169, 2003.

[6] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999.

[7] M. R. Fellows. New directions and new challenges in algorithm design and complexity, parameterized. In *Proc. of 8th WADS*, volume 2748 of LNCS, pages 505–519, Springer, 2003.

[8] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–30, 1997.

[9] J. Hromkovič. *Algorithmics for Hard Problems*, Second Edition. Springer, 2002.

[10] B. Monien. The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM J. Algebraic Discrete Methods*, 7: 505–512, 1986.

[11] R. Niedermeier. Ubiquitous parameterization—invitation to fixed-parameter algorithms. In *Proc. 29th MFCS*, volume 3153 of LNCS, pages 84–103, Springer, 2004.

[12] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, forthcoming, 2005.

[13] G. J. Woeginger. Exact algorithms for NP-hard problems: a survey. In *Proc. 5th International Workshop on Combinatorial Optimization*, volume 2570 of LNCS, pages 185–208, Springer, 2003.