Approximability of the Unsplittable Flow Problem on Trees*

Chrisil Arackaparambil Amit Chakrabarti Chien-Chung Huang

Department of Computer Science, Dartmouth College, Hanover, NH 03755, USA
{cja, ac, villars}@cs.dartmouth.edu

Dartmouth Computer Science Technical Report TR2009-642

Abstract

We consider the approximability of the Unsplittable Flow Problem (UFP) on tree graphs, and give a deterministic quasi-polynomial time approximation scheme for the problem when the number of leaves in the tree graph is at most poly-logarithmic in n (the number of demands), and when all edge capacities and resource requirements are suitably bounded. Our algorithm generalizes a recent technique that obtained the first such approximation scheme for line graphs. Our results show that the problem is not APX-hard for such graphs unless $\mathrm{NP} \subseteq \mathrm{DTIME}(2^{\mathrm{polylog}(n)})$. Further, a reduction from the Demand Matching Problem shows that UFP is APX-hard when the number of leaves is $\Omega(n^{\varepsilon})$ for any constant $\varepsilon > 0$.

Together, the two results give a nearly tight characterization of the approximability of the problem on tree graphs in terms of the number of leaves, and show the structure of the graph that results in hardness of approximation.

1 Introduction

In the Unsplittable Flow Problem (UFP) we are given as input a graph $G = (V, E)$ with edge capacities $\{c_e\}_{e \in E}$ and a set of n demands (numbered $1, \ldots, n$), the ith demand being specified by a 4-tuple (s_i, t_i, ρ_i, w_i) giving its source $s_i \in V$, its sink $t_i \in V$, its resource requirement ρ_i, and its profit w_i. A feasible solution is a subset $S \subseteq \{1, \ldots, n\}$ of the demands, along with routes from source to sink for each of the demands in S, so that, when they are routed simultaneously, no capacity constraint is violated. All of the resource requirement ρ_i of a demand $i \in S$ must be routed through the unique path chosen for it in the feasible solution. The objective now is to find a feasible solution S that maximizes the total profit $w(S) = \sum_{i \in S} w_i$. We assume that each c_e and ρ_i is an integer in $[1, L]$, where L is some appropriate bound, and that each w_i is an arbitrary integer.

This problem has a wide variety of applications including resource allocation, (throughput-maximizing) job scheduling, routing in networks, etc., to name a few. Many special cases of UFP are important and well-studied combinatorial optimization problems in their own right, with the Edge Disjoint Paths Problem being a major highlight: we refer the reader to the papers of Azar and Regev [AR01] and Chekuri, Mydlarz and Shepherd [CMS03], and the references therein, for details. Unfortunately, this means that UFP is a rather hard problem: even the very special case of a single-edge graph is NP-hard (equivalent to the Knapsack Problem). Of course, there are "degrees" of hardness, as measured by the best possible approximation factors obtainable in polynomial (or near-polynomial) time. Our goal in this work is to study the effect of the underlying graph structure on the approximability of a UFP instance.

It is intuitively clear that a more complex graph makes UFP a harder problem. The following table gives the supporting evidence, summarizing a number of results from previous work.

*Work supported in part by NSF grant EIA-98-02068.

Graph class	Approximability result	Hardness of approximation
General, directed	$O(\sqrt{\lvert E\rvert})$ [BS00, AR01]	$\Omega(\lvert E\rvert^{1/2-\varepsilon})$, unless $\mathrm{P} = \mathrm{NP}$ [GKR$^+$03]
Undirected	Same as above	$\Omega(\log^{1/2-\varepsilon} \lvert E\rvert)$ with no-bottleneck unless $\mathrm{NP} \subseteq \mathrm{ZPTIME}(2^{\mathrm{polylog}(n)})$ [ACKZ05]
		$\Omega(\lvert E\rvert^{1-\varepsilon})$ without no-bottleneck unless $\mathrm{P} = \mathrm{NP}$ [AR01]
Trees	$O(1)$ [CMS03] with no-bottleneck	$\Omega(1+\varepsilon)$, unless $\mathrm{P} = \mathrm{NP}$ [GVY97]
Stars	Same as above	$\Omega(1+\varepsilon)$, unless $\mathrm{P} = \mathrm{NP}$ [SV02]
Lines and cycles	Quasi-PTAS [BCES05]	NP-hard
	$O(\log n)$ [BFKS09]	
Single edge	FPTAS [IK75]	NP-hard

Table 1: Selected Highlights of Past Work on UFP

In this work, we seek to sharpen the boundary between graphs on which UFP is APX-hard and those on which it admits a quasi-polynomial time approximation scheme (QPTAS). (Note that the existence of a QPTAS implies that the problem is *not* APX-hard unless $\mathrm{NP} \subseteq \mathrm{DTIME}(2^{\mathrm{polylog}(n)})$.) Our results are.

Theorem 1. *There is a quasi-polynomial time approximation scheme for* UFP *on trees, provided the number of leaves is bounded by* $\mathrm{polylog}(n)$ *and all capacities and resource requirements are integers bounded by* $2^{\mathrm{polylog}(n)}$*, where* n *is the number of demands.*

Theorem 2. UFP *is APX-hard on trees with at least* $\Omega(n^{\varepsilon})$ *leaves, for any constant* $\varepsilon > 0$*, where* n *is the number of demands.*

We remark that Theorem 1 does not require the so-called "no-bottleneck assumption," that the maximum demand is no more than the minimum capacity. A number of algorithms for UFP developed in past research do require this assumption. Also, Theorem 2 depends on instances that violate this assumption.

Our Techniques: The main idea behind our algorithm is to generalize the ideas of Bansal et al. [BCES05], who recently obtained the first quasi-PTAS for UFP on line graphs. Their algorithm (called LINE-UFP-RECURSIVE), like ours, does not require a no-bottleneck assumption. It first divides the line graph into two halves at the combinatorial mid-point, "guesses" which of the demands crossing the mid-point should be selected, and then recursively solves the residual problem on each half. The guessing is done by first separating the demands crossing the mid-point into those that are "large" and those that are "small". The large demands in an optimal solution cannot be too many, so the algorithm performs an exhaustive search for the optimal subset. On the other hand, the capacity usage "profile" of the optimal subset of small demands can be approximated by a combinatorially simpler profile, and then a high-profit subset of the small demands can be packed into this profile by an algorithm called PILE-PACK that solves a linear program and rounds its solution. This profile approximation hinges on a simple, yet crucial, fact: there is a single edge e in the graph whose capacity is used by all of the demands that cross the mid-point. Following Bansal et al., we say that those demands form a *pile* at e.

The approximation by a simpler profile and the rounding result in a loss of an ε fraction of the optimal profit. The profile is guessed such that it never exceeds the capacity consumed in the optimal solution so that the recursive applications of the algorithm on the two halves have available at least the capacity required by the optimal.

Our algorithm first determines a number of *segments* in the graph. A segment is a maximal line subgraph that does not contain a non-leaf vertex of degree > 2 in the original graph. The main idea is that the LINE-UFP-RECURSIVE algorithm can be used on segments of a graph and further, demands that span multiple segments can be approximated "in parallel" by a suitable generalization of the techniques of Bansal et al. We first partition the demands into *zones*. We have a zone for each segment, which contains demands whose both end-points lie in that segment. We also have a zone for each pair of segments, which contains demands with an end-point in each segment. For each zone of the second type, we identify "large" and "small" demands. Then, we exhaustively search amongst the large demands and approximate the small demands in each zone in parallel, obtaining a $(1+\varepsilon)$-approximation. Demands in distinct intra-segment zones do not interact, so these can be handled by the LINE-UFP-RECURSIVE algorithm. On the other hand, demands in any one inter-segment zone form a pile at some suitable edge in the graph, so the approximating profile technique can be used.

The APX-hardness result is straightforward. Shepherd and Vetta [SV02] showed that the Demand Matching Problem (see Section 4 for a definition) is APX-hard. We give a simple reduction from this problem to UFP on a tree with $\Omega(n^{\varepsilon})$ leaves, for any $\varepsilon > 0$, thus showing that UFP is APX-hard on such trees. The connection between the two problems was already observed in [SV02]; the main new observation here is that the number of leaves can be made small as a function of n, the number of demands.

2 Preliminaries

We first discard all demands i with profit $w_i < (\delta/n)w_{\max}$, losing at most a δ fraction of the profit. We also scale the profits, resource requirements and capacities, so that $\rho_{\max} = w_{\min} = 1$. This gives us the bound $w_{\max} \leq n/\delta$, thereby ensuring that $1 \leq w_i/\rho_i \leq Ln/\delta$ for each demand i. We partition the set of demands into profit density classes: class-q contains demands i with $2^{q-1} \leq w_i/\rho_i < 2^q$. Accordingly, the previous bounds give us that there are at most $Q := 1 + \lfloor \lg \max_i\{w_i/\rho_i\} \rfloor \leq \text{polylog}(n)$ such classes, provided $L \leq 2^{\text{polylog(n)}}$. We can also discard vertices in the graph of degree ≤ 2 that are neither source nor sink. When a vertex of degree 2 is deleted, the two incident edges should be merged with a capacity equal to the minimum of the two edge capacities. After this preprocessing, the number of vertices of degree ≤ 2 is at most $2n$. We impose a total order on all edges in E. The *load* of a subset S' of demands on an edge e_i is defined as $\text{load}(S', e_i) := \sum_{j \in S', e_i \in s_j - t_j \text{path}} \rho_j$. A *profile* is a $|E|$-dimensional vector indexed by the edge order. For example, the edge capacities c form a profile. Another example is *resource profile* (or simply *profile*): given a subset of demands S', its resource requirement is $\text{prof}(S') := (\text{load}(S', e_1), \text{load}(S', e_2), \cdots, \text{load}(S', e_{|E|}))$. We use operators such as "$\leq$" and "$+$" on profiles in the standard coordinate-wise manner. For example, we can express the feasibility of S' by writing $\text{prof}(S') \leq c$.

Another important tool used in [BCES05] is the *δ-restricted profile* (also δ-RP). It is defined as follows.

Definition 1. Let $\overline{G}$ be a line subgraph of G consisting of consecutive vertices $\overline{v}_1, \overline{v}_2, \cdots, \overline{v}_k$. Furthermore, let $\overline{e}$ be an edge in $\overline{G}$ connecting $\overline{v}_{u-1}$ and $\overline{v}_u$, and h and δ be positive reals with $h \leq c_{\overline{e}}, \delta < 1$ and $1/\delta$ an integer. Let $x_1, \cdots, x_{1/\delta}$ and $y_1, \cdots, y_{1/\delta}$ be sequences of integers with $1 \leq x_1 \leq x_2 \leq$

$\cdots \leq x_{1/\delta} \leq u-1$ and $u \leq y_{1/\delta} \leq \cdots \leq y_2 \leq y_1 \leq k$. Then the profile $(l_1, \cdots, l_k)$, where

$$l_i = \begin{cases} 0, & \text{for } i \leq x_1 \text{ and } i > y_1 \\ j\delta h, & \text{for } x_j < i \leq x_{j+1} \text{ and } y_{j+1} < i \leq y_j \\ h, & \text{for } x_{1/\delta} < i \leq y_{1/\delta}, \end{cases}$$

is said to be a δ-restricted profile on $\overline{G}$ with peak $\overline{e}$ and height h, parameterized by the x_j's and y_j's.

On a line graph with m edges, given peak e and height h, a δ-restricted profile is completely determined by the choices of $x_1, \ldots, x_{1/\delta}$ and $y_1, \ldots, y_{1/\delta}$. Accordingly, there may be at most $m^{2/\delta}$ distinct profiles.

Definition 2 (Segment of a graph)**.** A segment of graph G is a maximal line subgraph G' of G such that the non-leaf vertices of G' do not have degree > 2 in the original graph G.

3 A Quasi-PTAS for the UFP

In this section we prove Theorem 1 by giving a Quasi-PTAS for UFP on trees with an appropriately bounded number of leaves. We first describe the algorithm using pseudocode, and then give proofs of its running time and correctness. We use PILE-PACK to denote the polynomial time approximation routine from Bansal et al. [BCES05]: this routine packs a high-profit subset of "small" demands that form a pile at an edge e into a given δ-restricted profile with peak e.

For the sake of clarity the algorithm is divided into three subroutines. TREE-UFP is the main routine that performs some preprocessing and then calls the subroutine ACROSS-SEGMENT-PHASE, which in turn makes a call to the subroutine GUESS. TREE-UFP first computes the segments $G_i, 1 \leq i \leq r$ of the input graph G (Line 1) and intra-segment zone of demands D_{ii} with source and sink in segment G_i (Line 2). It then determines minimal line graphs G_{ij} containing a pair of segments for each pair $(G_i, G_j), i < j$ (Line 4). Line 5 computes inter-segment zone D_{ij} containing demands having an endpoint in each segment of the pair. Finally Line 6 identifies the edge e^*_{ij} on which demands in D_{ij} form a pile. These steps are easily implemented using Breadth First Search. ACROSS-SEGMENT-PHASE is the recursive subroutine that, in parallel finds an approximate selection of the demands in inter-segment zones. For each (i, j) the call to GUESS returns the set $\mathcal{X}$ of guesses for the demands from D_{ij} to be selected in the solution. ACROSS-SEGMENT-PHASE when called by TREE-UFP with the arguments $i = r$ and $j = r-1$ recurses on every possible combination of i, j for $1 \leq j < i \leq r$ (a total of $\binom{r}{2}$ possible combinations) for each set of guessed demands in $\mathcal{X}$. The set of demands returned by the recursive

Input: tree $G = (V, E)$, edge capacities $\{c_e : e \in E\}$, demand set D
Output: a subset of D that can be feasibly routed giving profit $\geq (1 - O(\delta))\cdot$OPT

1 find the segment-graphs $(G_1, G_2, \ldots, G_r)$ of graph G
2 **for** *i=1* **to** *r* **do** find demands $D_{ii} = \{k \in D : s_k, t_k \in V(G_i)\}$
3 **for** $1 \leq i < j \leq r$ **do**
4 find the minimal line graph G_{ij} including the segments G_i and G_j
5 find demands
$D_{ij} = \{k \in D : |\{s_k, t_k\} \cap (V(G_i) \setminus V(G_j))| = 1$ and $|\{s_k, t_k\} \cap (V(G_j) \setminus V(G_i))| = 1\}$
6 find the edge e^*_{ij} in G_{ij} that is part of G_i and closest to G_j
7 $S \leftarrow$ ACROSS-SEGMENT-PHASE$(r, r-1, c)$
8 **return** S

Algorithm 1: TREE-UFP (G, c, D)

if $i \leq 1$ *and* $j \leq 1$ **then**
  $S \leftarrow \emptyset$
  for *k=1* **to** *r* **do**
    $S' \leftarrow$ LINE-UFP-RECURSIVE(G_k, c, D_{kk})
    $S \leftarrow S \cup S'$
  return S
$\mathcal{X} \leftarrow$ GUESS(i, j, c)
foreach $(S', c') \in \mathcal{X}$ **do**
  if $j = 1$ **then**
    $S'' \leftarrow$ ACROSS-SEGMENT-PHASE$(i-1, i-2, c')$
  else
    $S'' \leftarrow$ ACROSS-SEGMENT-PHASE$(i, j-1, c')$
  record the solution $S' \cup S''$
return S, *the most profitable of the recorded solutions*

Algorithm 2: ACROSS-SEGMENT-PHASE (i, j, c)

$\mathcal{X} \leftarrow \emptyset$
for *q=1* **to** *Q* **do** $D_{ijq} \leftarrow \{d \in D_{ij} : 2^{q-1} \leq w_d/\rho_d < 2^q\}$
foreach $(T_1, T_2, \ldots, T_Q)$, *with* $T_q \subseteq D_{ijq}$ *and* $|T_q| \leq 1/\delta^2$ **do**
  $T \leftarrow \bigcup_{q=1}^{Q} T_q$
  if T *can be routed* **then**
    route T and update c to c', the residual capacity
    foreach $(h'_1, h'_2, \ldots, h'_Q) \in \mathbb{R}^Q$, *such that* $\rho_{\min}$ *divides* h'_q *and* $\sum_{q=1}^{Q} h'_q \leq c'_{e^*_{ij}}$ **do**
      for *q=1* **to** *Q* **do** $S_{q,\text{small}} \leftarrow \{d \in D_{ijq} \setminus T_q : \rho_d \leq \delta^2(\rho_{\min} + h'_q + \text{load}(T_q, e^*_{ij}))\}$
      foreach $(\pi_1, \ldots, \pi_Q)$ *with* π_q *a* δ*-RP in* G_{ij} *on* e^*_{ij} *with height* h'_q, $\sum_{q=1}^{Q} \pi_q \leq c'$ **do**
        for *q=1* **to** *Q* **do** $U_q \leftarrow$ PILE-PACK$(G_{ij}, \pi_q, S_{q,\text{small}})$
        $U \leftarrow \bigcup_{q=1}^{Q} U_q$
        route all demands in U and obtain residual capacities c''
        $\mathcal{X} \leftarrow \mathcal{X} \cup \{(T \cup U, c'')\}$
return $\mathcal{X}$

Algorithm 3: GUESS(i, j, c)

call is combined with the corresponding set of guessed demands from $\mathcal{X}$ and the resulting solution is recorded. The best of all the recorded solutions is finally returned. ACROSS-SEGMENT-PHASE escapes the recursion by finally using LINE-UFP-RECURSIVE to process the demands in intra-segment zones. The demands in different intra-segment zones do not overlap, enabling us to process them independently. No parallel processing is required here.

The guessing strategy used in GUESS is the same as that in LINE-UFP-RECURSIVE. First the demands are classified into $Q \leq$ polylog(n) profit density classes according to the profit density ratio w_d/ρ_d. Then, for each of these classes, the large demands are first selected by exhaustive search since their number is bounded. The small demands are approximated by a δ-restricted profile that is again chosen by searching exhaustively. The number of choices for the δ-restricted profile are limited given its

simple description. Finally, the available small demands are packed into the profile using the PILE-PACK routine.

We make the crucial observation that there exists a sequence of recursive calls (a "thread of parallel execution") in which the load of the selected demands does not exceed the optimal at each step in the sequence. So we will have sufficient capacity available on each edge in successive recursive calls in this thread. We are therefore able to compare our solution with the optimal and bound the profit loss due to the δ-restricted profile and PILE-PACK.

We now move on to the proofs of the running time of the algorithm and its correctness. It is not hard to see with a few examples that the number of leaves in a tree constrains the number of segments possible. The following lemma gives a bound on the number of segments in terms of the number of leaves.

Lemma 3. *If n_1 is the number of leaves in a tree $G = (V, E)$ with maximum degree $d > 0$, then the number of segments r in G is bounded by $r \leq 2n_1 - 3$.*

Proof. For $i \in [d]$ let n_i denote the number of vertices having degree i in G. We can delete all n_2 vertices of degree 2 in the graph by merging the edges adjacent to these vertices. Call the new graph $G' = (V', E')$. Note that the number of edges $|E'|$ in this graph is exactly the number of segments r in G' (as well as in G). By the Handshaking Lemma we have that $n_1 + \sum_{i=3}^{d} i \cdot n_i = 2|E'| = 2|V'| - 2 = 2(n_1 + \sum_{i=3}^{d} n_i) - 2$. By rearranging terms we get $2n_1 = 2 + n_1 + \sum_{i=3}^{d}(i-2) \cdot n_i \geq 2 + |V'| = 3 + |E'|$. So $r = |E'| \leq 2n_1 - 3$. □

Observe that in the above lemma, $|E| = |E'| + n_2 \leq 2n_1 + n_2 - 3$. We will need this inequality in the next proof.

Theorem 4. *Algorithm* TREE-UFP *runs in time quasi-polynomial in n, provided that the number of leaves is polylog(n) and that L is quasi-polynomial in n.*

Proof. The preprocessing in TREE-UFP can be implemented using Breadth First Search requiring time polynomial in n. As called by TREE-UFP, ACROSS-SEGMENT-PHASE runs $\binom{r}{2} + 1$ times. The last run calls LINE-UFP-RECURSIVE on each of the segment graphs which requires $r \cdot 2^{\text{polylog}(n)}$ running time. All other runs of ACROSS-SEGMENT-PHASE call GUESS.

To bound the running time of GUESS, note that each of the sets T_q is of size at most $1/\delta^2$, so that the loop of line 3 in GUESS iterates at most n^{Q/δ^2} times. Also, for the loop on line 7, note that $\frac{\max_{e \in E} c_e}{\rho_{\min}} \leq L$, so that the loop iterates at most L^Q times. If n_i are as in Lemma 3 then we made the observation that that $m \leq 2n_1 + n_2 - 3$. Also, we noted previously that $n_1 + n_2 \leq 2n$, so that the number of edges $m \leq 4n$. Finally, we know that there are at most $m^{2/\delta}$ choices for a δ-restricted profile π_q given edge e^*_{ij} and height h_q, so that the loop on line 9 iterates at most $m^{2Q/\delta} \leq (4n)^{2Q/\delta}$ times. Thus, the running time for any iteration of GUESS is bounded by $n^{Q/\delta^2} L^Q (4n)^{2Q/\delta}$ which is also the branching factor (number of guesses in $\mathcal{X}$). This expression is bounded by $2^{\text{polylog}(n)}$ given that L is quasi-polynomial in n.

Now, using Lemma 3, we have that r is polylog(n) since the number of leaves is polylog(n). So the total running time of TREE-UFP is bounded by $(2^{\text{polylog}(n)})^{\binom{r}{2}} \times r 2^{\text{polylog}(n)}$ which is quasi-polynomial in n. □

We now show that TREE-UFP gives us the required approximation guarantee. The proof of Lemma 5 follows the arguments of Theorem 4.2 of [BCES05], and also this theorem directly implies Lemma 6.

Lemma 5. *If $\mathcal{O}$ denotes an optimal solution and δ is a small enough positive real such that $1/\delta$ is an integer then the set of guesses $\mathcal{X}$ produced by* GUESS(i, j, c), *$1 \leq i < j \leq r$ must contain a guess (S, c') such that,*

1. $w(S) \geq (1 - 13\delta)w(\mathcal{O} \cap D_{ij})$

2. *prof*$(S) \leq$ *prof*$(\mathcal{O} \cap D_{ij})$

provided that prof$(\mathcal{O} \cap D_{ij}) \leq c$.

Lemma 6. *If $\mathcal{O}$ denotes an optimal solution and δ is a small enough positive real such that $1/\delta$ is an integer then* LINE-UFP-RECURSIVE(G_i, c, D_{ii}) *produces a set S' such that $w(S') \geq (1 - 13\delta)w(\mathcal{O} \cap D_{ii})$, provided that prof$(\mathcal{O} \cap D_{ii}) \leq c$.*

Theorem 7. *If $\mathcal{O}$ denotes an optimal solution and δ is a small enough positive real such that $1/\delta$ is an integer then* TREE-UFP *returns a feasible solution with profit at least $(1 - 13\delta)w(\mathcal{O})$.*

Proof. First, it can be seen that recursive calls to ACROSS-SEGMENT-PHASE correspond to a recursion tree. A path from the root to a leaf in this tree denotes a sequence of recursive calls with arguments $(i, j) = (r, r-1), \ldots (r, 1), (r-1, r-2) \ldots (1, 1)$. At any vertex in a path we have $|\mathcal{X}|$ guesses, and this number gives the branching factor at that vertex. The choice of a guess in $\mathcal{X}$ determines the next vertex in the path. We claim that there must exist a path $\mathcal{P}$ from the root to a leaf vertex, where at each vertex in the path, the guess (S_{ij}, c_{ij}) selected is such that $w(S_{ij}) \geq (1 - 13\delta)w(\mathcal{O} \cap D_{ij})$ and also prof$(S_{ij}) \leq$ prof$(\mathcal{O} \cap D_{ij})$. We can find such a path by the following inductive argument. Note that at the root, with $(i, j) = (r, r - 1)$, no capacity has been used, so the available capacity $c_{ij} \geq$ prof$(\mathcal{O} \cap D_{ij})$. Then, to select the next vertex in the path, we consider a guess (S_{ij}, c'_{ij}) such that $w(S_{ij}) \geq (1-13\delta)w(\mathcal{O} \cap D_{ij})$ and prof$(S_{ij}) \leq$ prof$(\mathcal{O} \cap D_{ij})$. The existence of such a guess is guaranteed by Lemma 5. Now, suppose that we have determined such a path $\mathcal{P}$ partially and the next vertex to be selected has $(i, j) = (i', j')$. Since at every vertex with $(i, j) \in \mathcal{P}$, prof$(S_{ij}) \leq$ prof$(\mathcal{O} \cap D_{ij})$, the set of demands $\cup_{(i,j)\in\mathcal{P}} S_{ij}$ accumulated along the path does not load any edge more than $\mathcal{O} \cap \cup_{(i,j)\in\mathcal{P}} D_{ij}$, the optimal restricted to the zones of demands along the path. That is, prof$(\cup_{(i,j)\in\mathcal{P}} S_{ij}) \leq$ prof$(\mathcal{O} \cap \cup_{(i,j)\in\mathcal{P}} D_{ij})$. This implies that $c_{i'j'} \geq$ prof$(\mathcal{O} \cap D_{i'j'})$, since the demand zones D_{ij} form a partition of the set of demands D. Now we again apply Lemma 5 to determine the next guess with the desired properties. This proves our claim.

Finally, by Lemma 6 we see that in the final call to ACROSS-SEGMENT-PHASE, we get a set of demands guaranteeing a $(1 - 13\delta)$ factor of the optimal profit for zones $D_{ii}, 1 \leq i \leq r$, using LINE-UFP-RECURSIVE. □

4 APX-Hardness of UFP on Trees

In this section, we show the hardness of approximating the UFP on trees.

Theorem 8. UFP *is APX-hard on trees, when the number of leaves is $\Omega(n^{\varepsilon})$, for any $\varepsilon > 0$.*

Proof. We present a reduction from the demand matching problem. In an instance of the demand matching problem, we are given an undirected graph $G = (V, E)$, with capacities $c(v)$ for vertices $v \in V$; moreover, each edge $e \in E$ has a profit w_e and a demand ρ_e. The objective is to find a subset $E' \subseteq E$ so that the total profit $\sum_{e\in E'} w_e$ is maximized, under the constraint that the total demand of the chosen edges incident on a vertex is no more than the capacity of the vertex, that is for any $v \in V$, $\sum_{e\in E'\cap\delta(v)} \rho_e \leq c(v)$, where $\delta(v)$ denotes the edges incident to v. This problem was proved to be APX-hard by Shepherd and Vetta [SV02].

The reduction is as follows. We create a star graph. Each vertex v in the given demand matching problem instance translates to an edge e_v in the derived instance. Such an edge has capacity c_{e_v} equal to the capacity $c(v)$ of the vertex. For each edge $e = (u, v)$ in the original instance, we create a demand in the derived instance. The source and destination are the two leaf nodes which are incident to the edges e_u and e_v.

It can be seen that a maximum profit subset of demands to be routed in the derived instance also gives an optimal demand matching in the original instance. Note that in the derived instance, the number of demands is $|E|$ while the number of leaves is $|V|$. Given the fact that $|E| \leq |V|^2$, we have that the number of leaves is $\Omega(n^{1/2})$, where $n = |E|$, the number of demands in the derived instance. To ensure that the number of leaves is $\Omega(n^{\varepsilon})$, we create two more leaves in the star graph and give the two incident edges infinite capacity. Moreover, we add $|V|^{1/\varepsilon}$ demands whose sources and sinks are at these two newly added leaves. After adding these "dummy" demands, we have the desired leaf/demands proportion. □

References

[ACKZ05] Matthew Andrews, Julia Chuzhoy, Sanjeev Khanna, and Lisa Zhang. Hardness of the undirected edge-disjoint paths problem with congestion. In *Proc. 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 226–244, 2005.

[AR01] Yossi Azar and Oded Regev. Strongly polynomial algorithms for the unsplittable flow problem. In *Proc. 8th Conference on Integer Programming and Combinatorial Optimization*, pages 15–29, 2001.

[BCES05] Nikhil Bansal, Amit Chakrabarti, Amir Epstein, and Baruch Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *Proc. 38th Annual ACM Symposium on Theory of Computing*, pages 721–729, 2006.

[BFKS09] Nikhil Bansal, Zachary Friggstad, Rohit Khandekar, and Mohammad R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 702–709, 2009.

[BS00] Alok Baveja and Aravind Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Math. Oper. Res.*, 25(2):255–280, 2000.

[CMS03] Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree. In *Proc. 30th International Colloquium on Automata, Languages and Programming*, pages 410–425, 2003.

[GKR$^+$03] Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, F. Bruce Shepherd, and Mihalis Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *J. Comput. Syst. Sci.*, 67(3):473–496, 2003. Preliminary version in *Proc. 31st Annual ACM Symposium on the Theory of Computing*, pages 19–28, 1999.

[GVY97] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

[IK75] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975.

[SV02] F. Bruce Shepherd and Adrian Vetta. The demand matching problem. In *Proc. 9th Conference on Integer Programming and Combinatorial Optimization*, pages 457–474, 2002.