Combinatorial Algorithms for the Unsplittable Flow Problem *

Yossi Azar † Oded Regev ‡

January 10, 2005

Abstract

We provide combinatorial algorithms for the unsplittable flow problem (UFP) that either match or improve the previously best results. In the UFP we are given a (possibly directed) capacitated graph with n vertices and m edges, and a set of terminal pairs each with its own demand and profit. The objective is to connect a subset of the terminal pairs each by a single flow path subject to the capacity constraints such that the total profit of the connected pairs is maximized. We consider three variants of the problem. First is the *classical* UFP in which the maximum demand is at most than the minimum edge capacity. It was previously known to have an $O(\sqrt{m})$ approximation algorithm; the algorithm is based on the randomized rounding technique and its analysis makes use of the Chernoff bound and the FKG inequality. We provide a combinatorial algorithm that achieves the same approximation ratio and whose analysis is considerably simpler. Second is the *extended* UFP in which some demands might be higher than edge capacities. Our algorithm for this case improves the best known approximation ratio. We also give a lower bound that shows that the extended UFP is provably harder than the classical UFP. Finally, we consider the *bounded* UFP in which the maximum demand is at most $\frac{1}{K}$ times the minimum edge capacity for some $K > 1$. Here we provide combinatorial algorithms that match the currently best known algorithms. All of ours algorithms are strongly polynomial and some can even be used in the online setting.

1 Introduction

We consider the unsplittable flow problem (UFP). We are given a directed or undirected graph $G = (V, E)$, $|V| = n$, $|E| = m$, a capacity function u on its edges and a set of l terminal pairs of vertices (s_j, t_j) with demand d_j and profit r_j. A feasible solution is a subset S of the terminal pairs and a single flow path for each such pair such that the capacity constraints are fully met. The objective is to maximize the total profit of the satisfied terminal pairs. The well-known problem of maximum edge disjoint paths (EDP) is the special case in which all demands, profits and capacities are equal to 1.

Previous results: As already shown in [9], the EDP (and hence the UFP) is NP-complete. An $O(\sqrt{m})$ approximation algorithm is known for the EDP [10] (for additional positive results see [18, 19]). In [20], Srinivasan presented an $O(\sqrt{m})$ approximation algorithm for the UFP under the assumption that all edge capacities are equal. His algorithm was based on the randomized rounding

*A preliminary version of this paper appears in the proceedings of IPCO'01, pp. 15-29.

†Department of Computer Science, Tel Aviv University, Tel-Aviv, 69978, Israel. E-Mail: azar@tau.ac.il. Research supported in part by grants from the Israel Science Foundation and from the European Commission.

‡Department of Computer Science, Tel Aviv University, Tel-Aviv, 69978, Israel.

technique and its analysis involved several important probabilistic tools, such as the Chernoff bound and the FKG inequality. Kolliopoulos and Stein [12] considered a more general case of the UFP; their only assumption was that $d_{max} \leq u_{min}$ (i.e., the maximal demand is at most the minimal edge capacity). This assumption is known as the 'no-bottleneck' assumption. We will refer to UFP with this assumption as the *classical* UFP. Their result is an $O(\sqrt{m}\log m)$ approximation algorithm; it is based on good rounding techniques for some packing integer programs. Finally, by extending [20], Baveja and Srinivasan [2] provided an $O(\sqrt{m})$ approximation algorithm for the classical UFP; this is still the best approximation factor known for the problem. As before, the analysis of their algorithm involves the Chernoff bound and the FKG inequality.

In an interesting paper, Guruswami et al. [7] presented an $O(\sqrt{m\log m}\log\log m)$ approximation algorithm for the classical UFP. While weaker than the aforementioned approximation guarantee given by [2], the significance of their result is that its analysis is considerably simpler: it is based on basic randomized rounding. In the same paper the authors also considered combinatorial algorithms. Surprisingly, under certain conditions, they were able to achieve approximation ratios which are only slightly worse than those given by randomized rounding. Specifically, under the assumption that d_{max}/d_{min} is at most polynomial, their algorithm is an $O(\sqrt{m}\log m\log\log m)$ approximation algorithm. Here, d_{max} and d_{min} denote the maximum and minimum demands respectively. This raised the following natural question: is it possible to match the $O(\sqrt{m})$ given by [2] with a combinatorial algorithm? In this paper, we answer this question in the affirmative.

Another interesting result presented in [7] is an almost matching lower bound of $\Omega(m^{1/2-\epsilon})$ for any $\epsilon > 0$ under the assumption that $P \neq NP$ (a weaker bound was obtained independently by Ma and Wang [16]). The bound is shown for the directed EDP and hence it also applies to the classical UFP. Showing a similar bound for the undirected case is still an important open question.

Let us now consider the general UFP without the no-bottleneck assumption (specifically, some demands might be higher than some capacities). We will refer to this case as the *extended* UFP. The results of [7] imply an $O(\sqrt{m\log m}\max\{\log\log m, \log(d_{max}/u_{min})\})$ approximation algorithm based on randomized rounding where u_{min} denotes the minimum edge capacity. Moreover, their combinatorial algorithm can be seen to yield an $O(\sqrt{m}\log m\log(d_{max}/d_{min}))$ approximation guarantee.

We now consider the special case of UFP in which $d_{max} \leq \frac{1}{K}u_{min}$ for $K > 1$. We will refer to this case as the K-bounded UFP. For $K = \Omega(\log n)$, a constant approximation is shown in [17] by using randomized rounding. For constant K, an approximation ratio of $O(n^{\frac{1}{K-1}})$ was shown in [2] by using randomized rounding; the same approximation ratio can be obtained from [1] by combinatorial methods (see [3]). An improved ratio of $O(n^{\frac{1}{K}})$ is obtained in [12] by randomized rounding. However, prior to this work, no matching combinatorial algorithm was known.

Our results: Our main result is a combinatorial algorithm that achieves the $O(\sqrt{m})$ approximation ratio of [2]. Our algorithm has the advantage that its analysis is substantially simpler. This improves the combinatorial algorithm of [7] by eliminating the $\log m\log\log m$ factor and, more importantly, removing the assumption that d_{max}/d_{min} is at most polynomial. Essentially, our result shows how a purely combinatorial algorithm can replace a randomized rounding based one.

For the extended UFP, we improve [7] and obtain a combinatorial $O(\sqrt{m}\log(d_{max}/u_{min}))$ approximation algorithm. Notice that both the $\log m$ factor and the dependence on d_{min} are gone. In addition, under the assumption that $P \neq NP$, we prove a lower bound for the extended UFP over directed graphs. Specifically, we show that unless $P = NP$, it is impossible to approximate the extended UFP better than $O(m^{1-\epsilon})$ for any $\epsilon > 0$. This proves that the extended UFP is strictly harder than the classical UFP.

For the K-bounded UFP, we obtain an $O(Kn^{\frac{1}{K}})$ combinatorial approximation algorithm. This matches the previously best approximation guarantee. We note that we can replace n with D where D is an upper bound on the length of the longest path ever used in an optimal solution (which is obviously at most n).

All of our algorithms are shown to be strongly polynomial. Although the same applies to some algorithms based on randomized rounding, in our case the strong polynomiality is easy to obtain. This can be seen as one advantage of the combinatorial algorithms. Another advantage is that the algorithms are versatile and can be used in other settings. Here, we consider the online setting in which the network is known in advance but requests arrive one by one and a decision has to be made without knowing which requests follow. We present online algorithms whose competitive ratio is only slightly worse than that of the off-line algorithms. We also show that one of our algorithms is optimal in the online setting by improving a lower bound of [1].

Let us conclude with a summary of the main results in this paper:

- Classical UFP ($d_{max} \leq u_{min}$) - Combinatorial strongly polynomial $O(\sqrt{m})$ approximation algorithm.
- Extended UFP (arbitrary d_{max}, u_{min}) - Combinatorial strongly polynomial $O(\sqrt{m}\log(2+\frac{d_{max}}{u_{min}}))$ approximation algorithm; A lower bound of $\Omega(m^{1-\epsilon})$ and of $\Omega(m^{\frac{1}{2}-\epsilon}\sqrt{\log(2+\frac{d_{max}}{u_{min}})})$ for directed graphs.
- Bounded UFP ($d_{max} \leq \frac{1}{K}u_{min}$) - Combinatorial strongly polynomial $O(Kn^{\frac{1}{K}})$ approximation algorithm.

Recent results: Finally, let us mention some recent results that appeared after the original publication of the current paper. Chekuri and Khanna [5] considered approximation ratios in terms of the number of nodes n (and not the number of edges m as before). They were able to provide $n^{1-\epsilon}$ approximation algorithms for EDP for some value of ϵ. Kolman [13] generalized their result to the UFP under the no-bottleneck assumption and the assumption that profits equal demands. Moreover, improvements of their result were recently obtained by Varadarajan and Venkataraman [21] and by Hajiaghayi and Leighton [8]. Another extensive line of work concerns algorithms for the UFP whose performance depends on the expansion of the underlying undirected graph. Among the recent results is the work of Kolman and Scheideler [14, 15] who showed an $O(\Delta\alpha^{-1}(u_{max}/u_{min})\log n)$ approximation algorithm where Δ is the maximum degree and α is the expansion of the graph. Later, Chakrabarti et al. [4] provided a capacity-independent bound of $O(\Delta\alpha^{-1}\log^2 n)$; this improves [15] for the case where u_{max}/u_{min} is large.

2 Notation

Let $G=(V,E)$, $|V|=n$, $|E|=m$, be a (possibly directed) graph and a capacity function $u: E \to \mathbb{R}^+$. An input request is a quadruple (s_j, t_j, d_j, r_j) where (s_j, t_j) is the source-sink terminal pair, d_j is the demand and r_j is the profit. The input is a set of the above quadruples for $j \in T = \{1, \ldots, l\}$. Let D be a bound on the length of any routing path; note that D is at most n.

We denote by u_{min} (u_{max}) the minimum (maximum) edge capacity in the graph. Similarly, we define d_{min}, d_{max}, r_{min} and r_{max} to be the minimum/maximum demand/profit among all input requests. We define two functions on sets of requests, $S \subseteq T$:

$$r(S) = \sum_{j \in S} r_j \ , \qquad d(S) = \sum_{j \in S} d_j .$$

A feasible solution is a subset $\mathcal{P} \subseteq T$ and a route P_j from s_j to t_j for each $j \in \mathcal{P}$ subject to the capacity constraints, i.e., the total demand routed through an edge is bounded by the its capacity. Some of our algorithms order the requests so we will usually denote by $L_j(e)$ the relative load of edge e after routing request j, that is, the sum of demands routed through e divided by $u(e)$. Without loss of generality, we assume that any single request can be routed. That is possible since we can just ignore unroutable requests. Note that this is not the $d_{max} \leq u_{min}$ assumption made in the classical UFP.

Before describing the various algorithms, we begin with a simple useful lemma:

Lemma 2.1 *Given a sequence $\{a_1, \ldots, a_n\}$, a non-increasing non-negative sequence $\{b_1, \ldots, b_n\}$ and two sets $X, Y \subseteq \{1, \ldots, n\}$, let $X^i = X \cap \{1, \ldots, i\}$ and $Y^i = Y \cap \{1, \ldots, i\}$. If for some α and for every $1 \leq i \leq n$*

$$\sum_{j \in X^i} a_j > \alpha \sum_{j \in Y^i} a_j$$

then

$$\sum_{j \in X} a_j b_j > \alpha \sum_{j \in Y} a_j b_j$$

Proof: Denote $b_{n+1} = 0$. Since $b_j - b_{j+1}$ is non-negative,

$$\begin{aligned} \sum_{j \in X} a_j b_j &= \sum_{i=1}^{n} (b_i - b_{i+1}) \sum_{j \in X^i} a_j \\ &> \alpha \sum_{i=1}^{n} (b_i - b_{i+1}) \sum_{j \in Y^i} a_j = \alpha \sum_{j \in Y} a_j b_j \end{aligned}$$

■

3 Algorithms for the UFP

3.1 Algorithm for the Classical UFP

In this section we present the combinatorial algorithm for the classical UFP (i.e., the case in which $d_{max} \leq u_{min}$). The algorithm's approximation ratio is the same as that of the currently best known algorithm. Later, we will see that this algorithm can be easily made strongly polynomial and that it can even be used in the extended case.

We partition the set of requests T into two disjoint sets. The first, T_1, consists of requests for which $d_j \leq u_{min}/2$. The rest of the requests are in T_2. For each request j and a given path P from s_j to t_j define

$$F(j, P) = \frac{r_j}{d_j \sum_{e \in P} \frac{1}{u(e)}},$$

a measure of the profit gained relative to the added network load.

Given a set of requests, we use simple bounds on the values of F. The lower bound, denoted α_{lb}, is defined as $\frac{r_{min}}{n}$ and is indeed a lower bound on $F(j, P)$ since P cannot be longer than n edges and the capacity of its edges must be at least d_j. The upper bound, denoted α_{ub}, is defined as $\frac{r_{max} u_{max}}{d_{min}}$ and is clearly an upper bound on $F(j, P)$.

PROUTE
run $Routine_2(T_1)$ and $Routine_2(T_2)$ and choose the better solution

$Routine_2(S)$:
for each k from $\lfloor \log \alpha_{lb} \rfloor$ to $\lceil \log \alpha_{ub} \rceil$
run $Routine_1(2^k, S)$ and choose the best solution

$Routine_1(\alpha, S)$:
sort the requests in S according to a non-increasing order of r_j/d_j
for each $j \in S$ in the above order
if $\exists$ path P from s_j to t_j s.t. $F(j,P) > \alpha$ and $\forall e \in P, L_{j-1}(e) + \frac{d_j}{u(e)} \leq 1$
then route the request on P and for $e \in P$ set $L_j(e) = L_{j-1}(e) + \frac{d_j}{u(e)}$
else reject the request

Theorem 3.1 *Algorithm PROUTE is an $O(\sqrt{m})$ approximation algorithm for the classical* UFP.

Proof: First, we look at the running time of the algorithm. The number of iterations done in $Routine_2$ is:

$$\log \frac{\alpha_{ub}}{\alpha_{lb}} = \log(n \frac{r_{max}}{r_{min}} \frac{u_{max}}{d_{min}})$$

which is polynomial. $Routine_1$ looks for a non overflowing path P with $F(j,P) > \alpha$. The latter condition is equivalent to $\sum_{e \in P} \frac{1}{u(e)} < \frac{r_j}{d_j \alpha}$ and thus a shortest path algorithm can be used.

Consider an optimal solution routing requests in $\mathcal{Q} \subseteq T$. For each $j \in \mathcal{Q}$ let Q_j be the route chosen for j in the optimal solution. The total profit of either $\mathcal{Q} \cap T_1$ or $\mathcal{Q} \cap T_2$ is at least $\frac{r(\mathcal{Q})}{2}$. Denote that set by $\mathcal{Q}'$ and its index by $i' \in \{1,2\}$, that is, $\mathcal{Q}' = \mathcal{Q} \cap T_{i'}$. Now consider the values given to α in $Routine_2$ and let $\alpha' = 2^{k'}$ be the highest such that $r(\{j \in \mathcal{Q}' \mid F(j,Q_j) > \alpha'\}) \geq r(\mathcal{Q})/4$. It is clear that such an α' exists. From now on we limit ourselves to $Routine_1(\alpha', i')$ and show that a good routing is obtained by it. Denote by $\mathcal{P}$ the set of requests routed by $Routine_1(\alpha', i')$ and for $j \in \mathcal{P}$ denote by P_j the path chosen for it.

Let $\mathcal{Q}'_{high} = \{j \in \mathcal{Q}' \mid F(j,Q_j) > \alpha'\}$ and $\mathcal{Q}'_{low} = \{j \in \mathcal{Q}' \mid F(j,Q_j) \leq 2\alpha'\}$ be sets of higher and lower 'quality' routes in $\mathcal{Q}'$. Note that the sets are not disjoint and that the total profit in each of them is at least $\frac{r(\mathcal{Q})}{4}$ by the choice of α'. From the definition of F,

$$\begin{aligned} r(\mathcal{Q}'_{low}) = \sum_{j \in \mathcal{Q}'_{low}} F(j,Q_j) \sum_{e \in Q_j} \frac{d_j}{u(e)} &\leq 2\alpha' \sum_{j \in \mathcal{Q}'_{low}} \sum_{e \in Q_j} \frac{d_j}{u(e)} \\ &\leq 2\alpha' \sum_{j \in \mathcal{Q}} \sum_{e \in Q_j} \frac{d_j}{u(e)} \\ &= 2\alpha' \sum_{e} \sum_{j \in \mathcal{Q} | e \in Q_j} \frac{d_j}{u(e)} \\ &\leq 2\alpha' \sum_{e} 1 = 2m\alpha' \end{aligned}$$

where the last inequality is true since an optimal solution cannot overflow an edge. Therefore,

$$r(\mathcal{Q}) \leq 8m\alpha'.$$

Now let $E_{heavy} = \{e \in E \mid L_l(e) \geq \frac{1}{4}\}$ be a set of the heavy edges after the completion of $Routine_1(\alpha', i')$. We consider two cases. The first is when $|E_{heavy}| \geq \sqrt{m}$. According to the description of the algorithm, $F(j, P_j) > \alpha'$ for every $j \in \mathcal{P}$. Therefore,

$$\begin{aligned} r(\mathcal{P}) &= \sum_{j \in \mathcal{P}} F(j, P_j) \sum_{e \in P_j} \frac{d_j}{u(e)} \\ &\geq \alpha' \sum_{j \in \mathcal{P}} \sum_{e \in P_j} \frac{d_j}{u(e)} \\ &= \alpha' \sum_{e} \sum_{j | e \in P_j} \frac{d_j}{u(e)} \\ &= \alpha' \sum_{e} L_l(e) \geq \frac{1}{4} \sqrt{m} \alpha' \end{aligned}$$

where the last inequality follows from the assumption that more than $\sqrt{m}$ edges are loaded more than fourth their capacity. By combining the two inequalities we get

$$\frac{r(\mathcal{Q})}{r(\mathcal{P})} \leq 32\sqrt{m} = O(\sqrt{m}),$$

which completes the first case.

From now on we consider the second case where $|E_{heavy}| < \sqrt{m}$. Denote by $\mathcal{R} = \mathcal{Q}'_{high} \setminus \mathcal{P}$. We compare the profit given by our algorithm to that found in $\mathcal{R}$ by using Lemma 2.1. Since $\frac{r_j}{d_j}$ is a non increasing sequence, it is enough to bound the total demand routed in the prefixes of the two sets. For that we use the notation $\mathcal{R}^k = \mathcal{R} \cap \{1, \ldots, k\}$ and $\mathcal{P}^k = \mathcal{P} \cap \{1, \ldots, k\}$ for $k = 1, \ldots, l$. For each request $j \in \mathcal{R}^k$ the algorithm cannot find any appropriate path. In particular, the path Q_j is not chosen. Since $j \in \mathcal{Q}'_{high}$, $F(j, Q_j) > \alpha'$ and therefore the reason the path is not chosen is that it overflows one of the edges. Denote that edge by e_j and by $E^k = \{e_j \mid j \in \mathcal{R}^k\}$.

Lemma 3.2 $E^k \subseteq E_{heavy}$

Proof: Let $e_j \in E^k$ be an edge with $j \in \mathcal{R}^k$, a request corresponding to it. We claim that when the algorithm fails to find a path for j, $L_j(e_j) \geq \frac{1}{4}$. For the case $i' = 1$, the claim is obvious since the demand $d_j \leq u_{min}/2$ and in particular, $d_j \leq u(e_j)/2$. Thus, the load of e_j must be higher than $u(e_j)/2$ for the path Q_j to overflow it. For the case $i' = 2$, we know that $u_{min}/2 < d_j \leq u_{min}$. In case $u(e_j) > 2u_{min}$, the only way to overflow it with demands of size at most $d_{max} \leq u_{min}$ is when the edge is loaded at least $u(e_j) - u_{min} \geq u(e_j)/2$. Otherwise, $u(e_j) \leq 2u_{min}$ and since $d_j \leq u_{min} \leq u(e)$ we know that the edge cannot be empty. Since we only route requests from T_2 the edge's load must be at least $u_{min}/2 \geq u(e_j)/4$. ∎

Since each request in $\mathcal{R}^k$ is routed through an edge of E^k in the optimal solution, $d(\mathcal{R}^k) \leq \sum_{e \in E^k} u(e)$. The highest capacity edge $f \in E^k$ is loaded more than fourth its capacity since it is in E_{heavy} and therefore $d(\mathcal{P}^k) \geq \frac{u(f)}{4}$. By Lemma 3.2, $|E^k| \leq |E_{heavy}| < \sqrt{m}$ and hence,

$$d(\mathcal{R}^k) < \sqrt{m} \cdot u(f) \leq 4\sqrt{m} \cdot d(\mathcal{P}^k).$$

We use Lemma 2.1 by combining the inequality above on the ratio of demands and the nonincreasing sequence $\frac{r_j}{d_j}$. This yields

$$\sum_{j \in \mathcal{R}} \frac{r_j}{d_j} d_j \leq 4\sqrt{m} \sum_{j \in \mathcal{P}} \frac{r_j}{d_j} d_j,$$

or,

$$r(\mathcal{R}) \leq 4\sqrt{m} \cdot r(\mathcal{P}).$$

Since $\mathcal{Q}'_{high} \subseteq \mathcal{R} \cup \mathcal{P}$,

$$r(\mathcal{Q}'_{high}) \leq r(\mathcal{R}) + r(\mathcal{P}) \leq (1 + 4\sqrt{m})r(\mathcal{P}).$$

Recall that $r(\mathcal{Q}'_{high}) \geq r(\mathcal{Q})/4$ and therefore

$$\frac{r(\mathcal{Q})}{r(\mathcal{P})} \leq 4 + 16\sqrt{m} = O(\sqrt{m}).$$

■

3.2 Strongly Polynomial Algorithm

$Routine_1$ is strongly polynomial. $Routine_2$ however calls it $\log \frac{\alpha_{ub}}{\alpha_{lb}}$ times. Therefore, it is polynomial but still not strongly polynomial. We add a preprocessing step whose purpose is to bound the ratio $\frac{\alpha_{ub}}{\alpha_{lb}}$. Recall that l denotes the number of requests.

$SPROUTE(T)$:
  run $Routine_3(T_1)$ and $Routine_3(T_2)$ and choose the better solution

$Routine_3(S)$:
  For each edge such that $u(e) > l \cdot d_{max}$ set $u(e)$ to be $l \cdot d_{max}$.
  Throw away requests whose profit is below $\frac{r_{max}}{l}$.
  Take the better out of the following two solutions:
    Route all requests in $S_{tiny} = \{j \in S \mid d_j \leq \frac{u_{min}}{l}\}$ on any simple path.
    Run $Routine_2(S \setminus S_{tiny})$.

Theorem 3.3 *Algorithm $SPROUTE$ is a strongly polynomial $O(\sqrt{m})$ approximation algorithm for the classical* UFP.

Proof: Consider an optimal solution routing requests in $\mathcal{Q} \subseteq S$. Since the demand of a single request is at most d_{max}, the total demand routed through a given edge is at most $l \cdot d_{max}$. Therefore, $\mathcal{Q}$ is still routable after the first preprocessing phase. The total profit of requests whose profit is lower than $\frac{r_{max}}{l}$ is r_{max}. In case $r(\mathcal{Q}) > 2r_{max}$, removing these requests still leaves the set $\mathcal{Q}'$ whose total profit is at least $r(\mathcal{Q}) - r_{max} \geq \frac{r(\mathcal{Q})}{2}$. Otherwise, we take $\mathcal{Q}'$ to be the set containing the request of highest profit. Then, $r(\mathcal{Q}')$ is $r_{max} \geq \frac{r(\mathcal{Q})}{2}$. All in all, after the two preprocessing phases we are left with an UFP instance for which there is a solution $\mathcal{Q}'$ whose profit is at least $\frac{r(\mathcal{Q})}{2}$.

Assume that the total profit in $\mathcal{Q}' \cap S_{tiny}$ is at least $\frac{r(\mathcal{Q})}{4}$. Since the requests in S_{tiny} have a demand of at most $\frac{u_{min}}{l}$ and there are at most l of them, they can all be routed on simple paths and the profit obtained is at least $\frac{r(\mathcal{Q})}{4}$. Otherwise, the profit in $\mathcal{Q}' \setminus S_{tiny}$ is at least $\frac{r(\mathcal{Q})}{4}$ and since algorithm $PROUTE$ is an $O(\sqrt{m})$ approximation algorithm, the profit we obtain is also within $O(\sqrt{m})$ of $r(\mathcal{Q})$.

The preprocessing phases by themselves are obviously strongly polynomial. Recall that the number of iterations performed by $Routine_2$ is $\log(n\frac{r_{max}}{r_{min}}\frac{u_{max}}{d_{min}})$. The ratio of profits is at most l by the second preprocessing phase. The first preprocessing phase limits u_{max} to $l \cdot d_{max}$. So, the number of iterations is at most $\log(nl^2\frac{d_{max}}{d_{min}})$. In case $S = T_1$, $d_{max} \leq \frac{u_{min}}{2}$ and $d_{min} \geq \frac{u_{min}}{l}$ since tiny requests are removed. For $S = T_2$, $d_{max} \leq u_{min}$ and $d_{min} \geq u_{min}/2$. We end up with at most $O(\log n + \log l)$ iterations which is strongly polynomial. ■

3.3 Algorithm for the Extended UFP

In this section we show that the algorithm can be used for the extended case in which demands can be higher than the lowest edge capacity.

Instead of using just two sets in $SPROUTE$, we define a partition of the set of requests T into $2+\max\{\lceil \log d_{max}/u_{min} \rceil, 0\}$ disjoint sets. The first, T_1, consists of requests for which $d_j < u_{min}/2$. The set T_i for $i > 1$ is of requests for which $2^{i-3}u_{min} < d_j \leq 2^{i-2}u_{min}$. The algorithm is as follows:

$ESPROUTE(T)$:
let Z be $\{i \mid T_i \neq \emptyset\}$
for each $i \in Z$
  run $Routine_3(T_i)$ on the resulting graph
choose the best solution obtained

Theorem 3.4 *Algorithm $ESPROUTE$ is a strongly polynomial $O(\sqrt{m}\log(2+\frac{d_{max}}{u_{min}}))$ approximation algorithm for the extended* UFP.

Proof: We choose i' as the index that maximizes the profit in $\mathcal{Q} \cap T_{i'}$ and the proof essentially follows the proofs of Theorem 3.1 and Theorem 3.3. The only part which has to be modified is Lemma 3.2. The following lemma replaces it:

Lemma 3.5 $E^k \subseteq E_{heavy}$

Proof: Let $e_j \in E^k$ be an edge with $j \in \mathcal{R}^k$, a request corresponding to it. We claim that when the algorithm fails to find a path for j, $L_j(e_j) \geq \frac{1}{4}$. For the case $i' = 1$, the claim is obvious as before. For the case $i' > 1$, we know that $2^{i'-3}u_{min} < d_j \leq 2^{i'-2}u_{min}$. In case $u(e_j) > 2^{i'-1}u_{min}$, the only way to overflow it with demands of size at most $2^{i'-2}u_{min}$ is when the edge is loaded at least $u(e_j) - 2^{i'-2}u_{min} \geq u(e_j)/2$. Otherwise, $u(e_j) \leq 2^{i'-1}u_{min}$ and since j is routed through this edge in the optimal solution $d_j \leq u(e_j)$. Therefore, the edge cannot be empty. Since we only route requests from $T_{i'}$ the edge's load must be at least $2^{i'-3}u_{min} \geq u(e_j)/4$. ■

We now analyze the running time of the algorithm. Note that $|Z| \leq l$ and hence $ESPROUTE$ calls $Routine_3$ at most l times. Moreover, the set Z can be computed directly from T without having to go through all possible indices i (namely, for each request j compute the i for which $2^{i-3}u_{min} < d_j \leq 2^{i-2}u_{min}$ and add this i to Z). For T_1, the number of iterations of $Routine_2$ is the same as in $SPROUTE$. For a set T_i, $i > 1$, the number of iterations of $Routine_2$ is $\log(n\frac{r_{max}}{r_{min}}\frac{u_{max}}{d_{min}})$. As before, the preprocessing of $Routine_3$ reduces this number to $\log(nl^2\frac{d_{max}}{d_{min}})$. Since the ratio $\frac{d_{max}}{d_{min}}$ is at most 2 in each T_i, we conclude that $ESPROUTE$ is strongly polynomial.

■

4 Algorithms for the K-bounded UFP

In the previous sections we considered the classical and the extended UFP. In this section we present better algorithms for the K-bounded UFP in which $d_{max} \leq \frac{1}{K}u_{min}$ for some $K \geq 2$. Our algorithms are based on the exponential scale technique presented in [1].

4.1 Algorithms for Bounded Demands

In this section we present two algorithms for the bounded UFP. The first deals with the case in which the demands are in the range $[\frac{u_{min}}{K+1}, \frac{u_{min}}{K}]$. As a special case, it provides an $O(\sqrt{n})$ approximation algorithm for the half-disjoint paths problem where edge capacities are all the same and the demands are exactly half the edge capacity. The second is an algorithm for the K-bounded UFP where demands are only bounded by $\frac{u_{min}}{K}$ from above. Recall that D is an upper bound on the length of any routing path and in particular, $D \leq n$.

$EKROUTE(T)$:
  $\mu \leftarrow 2D$
  sort the requests in T according to a non-increasing order of r_j/d_j
  for each $j \in T$ in the above order
    if $\exists$ a path P from s_j to t_j s.t.
      $\sum_{e \in P}(\mu^{L_{j-1}(e)} - 1) < D$
    then route the request on P and for $e \in P$ set $L_j(e) = L_{j-1}(e) + \frac{1}{\lfloor \frac{K \cdot u(e)}{u_{min}} \rfloor}$
    else reject the request

$BKROUTE(T)$:
  $\mu \leftarrow (2D)^{1+\frac{1}{K-1}}$
  sort the requests in T according to a non-increasing order of r_j/d_j
  for each $j \in T_i$ in the above order
    if $\exists$ a path P from s_j to t_j s.t.
      $\sum_{e \in P}(\mu^{L_{j-1}(e)} - 1) < D$
    then route the request on P and for $e \in P$ set $L_j(e) = L_{j-1}(e) + \frac{d_j}{u(e)}$
    else reject the request

Note that algorithm $EKROUTE$ uses a slightly different definition of L. This 'virtual' relative load allows it to outperform $BKROUTE$ on instances where the demands are in the correct range.

Theorem 4.1 *Algorithm $EKROUTE$ is a strongly polynomial $O(K \cdot D^{\frac{1}{K}})$ approximation algorithm for the* UFP *with demands in the range $[\frac{u_{min}}{K+1}, \frac{u_{min}}{K}]$. Algorithm $BKROUTE$ is a strongly polynomial $O(K \cdot D^{\frac{1}{K-1}})$ approximation algorithm for the K-bounded* UFP.

Proof: The first thing to note is that the algorithms never overflow an edge. For the first algorithm, the demands are at most $\frac{u_{min}}{K}$ and the only way to exceed an edge capacity is to route request j through an edge e that holds at least $\lfloor \frac{K \cdot u(e)}{u_{min}} \rfloor$ requests. For such an edge, $L_{j-1}(e) \geq 1$ and $\mu^{L_{j-1}(e)} - 1 \geq \mu - 1 \geq D$. For the second algorithm, it is sufficient to show that in case $L_{j-1}(e) > 1 - \frac{1}{K}$ for some e then $\mu^{L_{j-1}(e)} - 1 \geq D$; that is true since $\mu^{L_{j-1}(e)} - 1 \geq ((2D)^{1+\frac{1}{K-1}})^{1-\frac{1}{K}} - 1 = 2D - 1 \geq D$. Therefore, the algorithms never overflow an edge.

Now we lower bound the total demand accepted by our algorithms. We denote by $\mathcal{Q}$ the set of requests in the optimal solution and by $\mathcal{P}$ the requests accepted by either of our algorithms. For $j \in \mathcal{Q}$ denote by Q_j the path chosen for it in the optimal solution and for $j \in \mathcal{P}$ let P_j be the path chosen for it by our algorithm. We consider prefixes of the input so let $\mathcal{Q}^k = \mathcal{Q} \cap \{1, \ldots, k\}$ and $\mathcal{P}^k = \mathcal{P} \cap \{1, \ldots, k\}$ for $k = 1, \ldots, l$. We prove that

$$d(\mathcal{P}^k) \geq \frac{\sum_e u(e)(\mu^{L_k(e)} - 1)}{6KD\mu^{\frac{1}{K}}}.$$

The proof is by induction on k and the induction base is trivial since the above expression is zero. Thus, it is sufficient to show that for an accepted request j

$$\frac{\sum_{e\in P_j} u(e)(\mu^{L_j(e)} - \mu^{L_{j-1}(e)})}{6KD\mu^{\frac{1}{K}}} \leq d_j.$$

Note that for any $e \in P_j$, $L_j(e) - L_{j-1}(e) \leq \frac{1}{K}$ for both algorithms. In addition, for both algorithms $L_j(e) - L_{j-1}(e) \leq 3\frac{d_j}{u(e)}$ where the factor 3 is only necessary for $EKROUTE$ where the virtual load is higher than the actual increase in relative load. The worst case is when $K = 2$, $u(e) = (1.5-\epsilon)u_{min}$ and $d_j = (\frac{1}{3} + \epsilon)u_{min}$: the virtual load increases by $\frac{1}{2}$ whereas $\frac{d_j}{u(e)}$ is about $\frac{2}{9}$. Looking at the exponent,

$$\begin{aligned}
\mu^{L_j(e)} - \mu^{L_{j-1}(e)} &= \mu^{L_{j-1}(e)}(\mu^{L_j(e)-L_{j-1}(e)} - 1) \\
&= \mu^{L_{j-1}(e)}((\mu^{\frac{1}{K}})^{K(L_j(e)-L_{j-1}(e))} - 1) \\
&\leq \mu^{L_{j-1}(e)}\mu^{\frac{1}{K}}K(L_j(e) - L_{j-1}(e)) \\
&\leq \mu^{L_{j-1}(e)}\mu^{\frac{1}{K}}3K\frac{d_j}{u(e)}
\end{aligned}$$

where the first inequality is due to the simple relation $x^y - 1 \leq xy$ for $0 \leq y \leq 1, 0 \leq x$ and that for $e \in P_j$, $L_j(e) - L_{j-1}(e) \leq \frac{1}{K}$. Therefore,

$$\begin{aligned}
\sum_{e\in P_j} u(e)(\mu^{L_j(e)} - \mu^{L_{j-1}(e)}) &\leq \sum_{e\in P_j} \mu^{L_{j-1}(e)}\mu^{\frac{1}{K}}3Kd_j \\
&= 3K\mu^{\frac{1}{K}}d_j \sum_{e\in P_j} \mu^{L_{j-1}(e)} \\
&= 3K\mu^{\frac{1}{K}}d_j(\sum_{e\in P_j} (\mu^{L_{j-1}(e)} - 1) + |P_j|) \\
&\leq 3K\mu^{\frac{1}{K}}(D + D)d_j \\
&= 6KD\mu^{\frac{1}{K}}d_j
\end{aligned}$$

where the last inequality holds since the algorithm routes the request through P_j and the length of P_j is at most D.

The last step in the proof is to upper bound the total demand accepted by an optimal algorithm. Denote the set of requests rejected by our algorithm and accepted by the optimal one by $\mathcal{R}^k = \mathcal{Q}^k \setminus \mathcal{P}^k$. For $j \in \mathcal{R}^k$, we know that $\sum_{e\in Q_j}(\mu^{L_{j-1}(e)} - 1) \geq D$ since the request is rejected by our algorithm. Hence,

$$\begin{aligned}
D \cdot d(\mathcal{R}^k) &\leq \sum_{j\in\mathcal{R}^k} \sum_{e\in Q_j} d_j(\mu^{L_{j-1}(e)} - 1) \\
&\leq \sum_{j\in\mathcal{R}^k} \sum_{e\in Q_j} d_j(\mu^{L_k(e)} - 1) \\
&= \sum_{e} \sum_{j\in\mathcal{R}^k | e\in Q_j} d_j(\mu^{L_k(e)} - 1) \\
&= \sum_{e} (\mu^{L_k(e)} - 1) \sum_{j\in\mathcal{R}^k | e\in Q_j} d_j \\
&\leq \sum_{e} (\mu^{L_k(e)} - 1)u(e),
\end{aligned}$$

where the last inequality holds since the optimal algorithm cannot overflow an edge.

By combining the two inequalities shown above,

$$d(\mathcal{Q}^k) \leq d(\mathcal{P}^k) + d(\mathcal{R}^k) \leq d(\mathcal{P}^k) + d(\mathcal{P}^k)\frac{6KD}{D}\mu^{\frac{1}{K}} = (1 + 6K\mu^{\frac{1}{K}})d(\mathcal{P}^k).$$

The algorithm followed a non-increasing order of $\frac{r_j}{d_j}$ and by Lemma 2.1 we obtain the same inequality above for profits. So, the approximation ratio of the algorithm is

$$1 + 6K\mu^{\frac{1}{K}} = O(K \cdot \mu^{\frac{1}{K}}),$$

which, by assigning the appropriate values of μ, yields the desired results. ■

4.2 A Combined Algorithm

In this section we combine the two algorithms presented in the previous section: the algorithm for demands in the range $[\frac{u_{min}}{K+1}, \frac{u_{min}}{K}]$ and the algorithm for the K-bounded UFP. The result is an algorithm for the K-bounded UFP with an approximation ratio of $O(K \cdot D^{\frac{1}{K}})$.

We define a partition of the set of requests T into two sets. The first, T_1, includes all the requests whose demand is at most $\frac{1}{K+1}$. The second, T_2, includes all the requests whose demand is more than $\frac{1}{K+1}$ and at most $\frac{1}{K}$.

$CKROUTE(T)$:
  Take the best out of the following two possible solutions:
    Route T_1 by using $BKROUTE$ and reject all requests in T_2
    Route T_2 by using $EKROUTE$ and reject all requests in T_1

Theorem 4.2 *Algorithm $CKROUTE$ is a strongly polynomial $O(K \cdot D^{\frac{1}{K}})$ approximation algorithm for the K-bounded* UFP.

Proof: Let $\mathcal{Q}$ denote an optimal solution in T. Since $BKROUTE$ is used with demands bounded by $\frac{1}{K+1}$ its approximation ratio is $O(KD^{\frac{1}{K}})$. The same approximation ratio is given by $EKROUTE$. Either T_1 or T_2 have an optimal solution whose profit is at least $\frac{r(\mathcal{Q})}{2}$ and therefore we obtain the claimed approximation ratio. ■

5 Lower Bounds

In this section we show that in cases where the demands are much larger than the minimum edge capacity, the UFP becomes very hard to approximate, namely, $\Omega(m^{1-\epsilon})$ for any $\epsilon > 0$. We also show how different demand values relate to the approximability of the problem. The lower bounds are for directed graphs only.

Theorem 5.1 *[6] The following problem is NPC:*

2DIRPATH:
INPUT: A directed graph $G = (V, E)$ and four nodes $x, y, z, w \in V$
QUESTION: Are there two edge disjoint directed paths,
one from x to y and the other from z to w in G ?

Theorem 5.2 *For any $\epsilon > 0$, the extended* UFP *cannot be approximated better than $\Omega(m^{1-\epsilon})$.*

Proof: For a given instance A of $2DIRPATH$ with $|A|$ edges and a small constant ϵ, we construct an instance of the extended UFP composed of l copies of A, $A^1, A^2, \ldots, A^l$ where $l = |A|^{\lceil \frac{1}{\epsilon} \rceil}$. The instance A^i is composed of edges of capacity 2^{l-i}. A special node y^0 is added to the graph. Two edges are added for each A^i, (y^{i-1}, x^i) of capacity $2^{l-i} - 1$ and (y^{i-1}, z^i) of capacity 2^{l-i}. All l requests share y^0 as a source node. The sink of request $1 \leq i \leq l$ is w^i. The demand of request i is 2^{l-i} and its profit is 1. This is illustrated in Figure 1 for the case $l = 4$. Each diamond indicates a copy of A with x, y, z, w being its left, right, top and bottom corners respectively. The number inside each diamond indicates the capacity of A's edges in this copy.

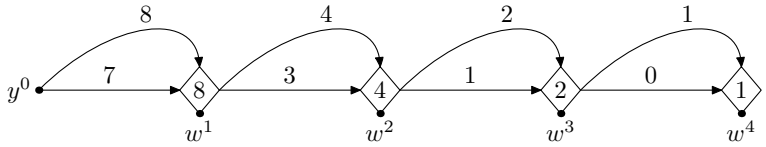

Figure 1: The UFP instance for the case $l = 4$

We claim that for a given YES instance of $2DIRPATH$ the maximal profit gained from the extended UFP instance is l. We route request $1 \leq i \leq l$ through $[y^0, x^1, y^1, x^2, y^2, \ldots, y^{i-1}, z^i, w^i]$. Note that the path from x^j to y^j and from z^j to w^j is a path in A^j given by the YES instance.

For a NO instance, we claim that at most one request can be routed. That is because the path chosen for a request i ends at w^i. So, it must arrive from either z^i or x^i. The only edge entering x^i is of capacity $2^{l-i} - 1$ so z^i is the only option. The instance A^i is a NO instance of capacity 2^{l-i} through which a request of demand 2^{l-i} is routed form z^i to w^i. No other path can therefore be routed through A^i so requests $j > i$ are not routable. Since i is arbitrary, we conclude that at most one request can be routed through the extended UFP instance and that its profit is 1.

The gap created is $l = |A|^{\frac{1}{\epsilon}}$ and the number of edges is $l \cdot (|A| + 2) = O(l^{1+\epsilon})$. Hence, the gap is $\Omega(m^{\frac{1}{1+\epsilon}}) = \Omega(m^{1-\epsilon'})$ and since ϵ is arbitrary we complete the proof. ■

Theorem 5.3 *For any $\epsilon > 0$, the extended* UFP *with any ratio $d_{max}/u_{min} \geq 2$ cannot be approximated better than $\Omega(m^{\frac{1}{2}-\epsilon}\sqrt{\lfloor \log(\frac{d_{max}}{u_{min}}) \rfloor})$.*

Proof: For a given instance A of $2DIRPATH$ with $|A|$ edges and a small constant ϵ, we construct an instance of the extended UFP with the given ratio d_{max}/u_{min}. We begin with describing our basic building block from which the UFP instance will be built. Let k, l be some parameters to be chosen later and let $\delta < \frac{1}{lk}$ be a small constant. The ith building block, denoted B^i, is illustrated in Figure 2. It contains $k^2 + \frac{k(k-1)}{2}$ copies of the graph A with edge capacity $2^{l-i} + k\delta$. For any two integers $1 \leq a \leq k$ and $1 \leq b \leq 2k - a$ there is a copy of A located at position (a, b) on a two-dimensional grid. There are $2k$ input nodes and $2k$ output nodes. The first k input nodes x^i_j, $1 \leq j \leq k$, are located at $(0, j)$ whereas the next k input nodes z^i_j, $1 \leq j \leq k$, are at $(0, k + j)$. The output nodes y^i_j, $1 \leq j \leq k$, are located at $(k + 1, j)$ and w^i_j, $1 \leq j \leq k$, are at $(k - j + 1, 0)$. The edges are described by their location on the grid and are all of length one on that grid. A copy of A located at (a, b) is connected through its z node to an edge above it (the edge from $(a, b + 1)$ to (a, b)), through its w node to the edge below it, through x to the left edge and through y to the right edge. For each $1 \leq j \leq k$ there exist $k + 1$ horizontal edges,

$$((0, j), (1, j)), \ ((1, j), (2, j)), \ \ldots, \ ((k, j), (k + 1, j)).$$

The capacity of each of these edges is 2^{l-i}. For $1 \leq j \leq k$ we also add $k-j+1$ horizontal edges

$$((0,k+j),(1,k+j)),\ \ldots,\ ((k-j,k+j),(k-j+1,k+j))$$

and $k+j$ vertical edges

$$((k-j+1,k+j),(k-j+1,k+j-1)),\ \ldots,\ ((k-j+1,1),(k-j+1,0)).$$

The capacity of each of these edges is $2^{l-i}+j\delta$.

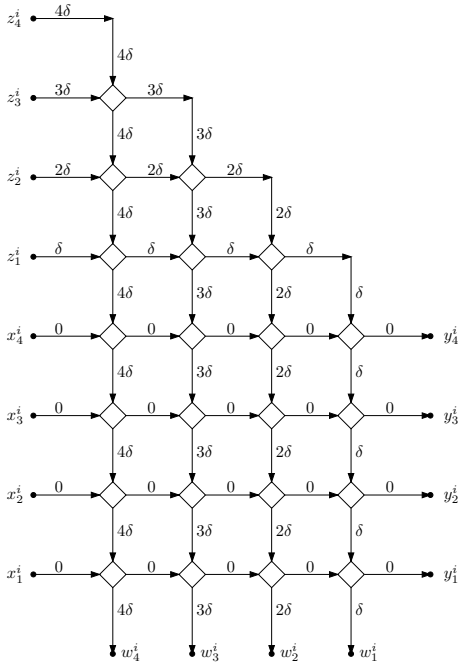

Figure 2: B^i for the case $k=4$. Each diamond corresponds to a copy of A with edge capacity $2^{l-i}+k\delta$. Edge capacities are 2^{l-i} plus the indicated amount.

The UFP instance consists of the l blocks B^1, B^2,...,B^l and is illustrated in Figure 3. There are k additional nodes denoted y_j^0, $1 \leq j \leq k$, which act as input nodes. For $1 \leq i \leq l$, $2k$ connecting edges of capacity 2^l are used, (y_j^{i-1}, x_j^i) and (y_j^{i-1}, z_j^i), $1 \leq j \leq k$. The request set consists of requests denoted $r_{(i,j)}$ for $1 \leq i \leq l, 1 \leq j \leq k$ from y_j^0 to w_j^i with demand $2^{l-i}+j\delta$. All requests are of profit 1.

We claim that for a given YES instance of $2DIRPATH$ the maximal profit gained from the extended UFP instance is $l \cdot k$, that is, all requests can be routed. We route request $r_{(i,j)}$ through $[y_j^0, x_j^1, y_j^1, x_j^2, y_j^2, \ldots, y_j^{i-1}, z_j^i, w_j^i]$. Note that the path from x_j^i to y_j^i is a horizontal path in B^i through k copies of A and the path from z_j^i to w_j^i is a path in B^i going through the point $(k-j+1, k+j)$ and passing $2k-1$ copies of A. The load in B^i on the edges of the path from x_j^i to y_j^i is $2^{l-i-1}+j\delta+2^{l-i-2}+j\delta+\ldots+1+j\delta \leq 2^{l-i}-1+lk\delta \leq 2^{l-i}$. The load in B^i on the edges of the path from z_j^i to w_j^i is $2^{l-i}+j\delta$. Therefore, no edge is overloaded and the total profit gained is as claimed.

For a NO instance, we claim that at most one request can be routed. Assume request $r_{(i,j)}$ is routed through the UFP instance. Request $r_{(i,j)}$ is of demand $2^{l-i}+j\delta$ and by tracing back the request from its sink w_j^i it can be seen that the capacities of the edges are such that the request

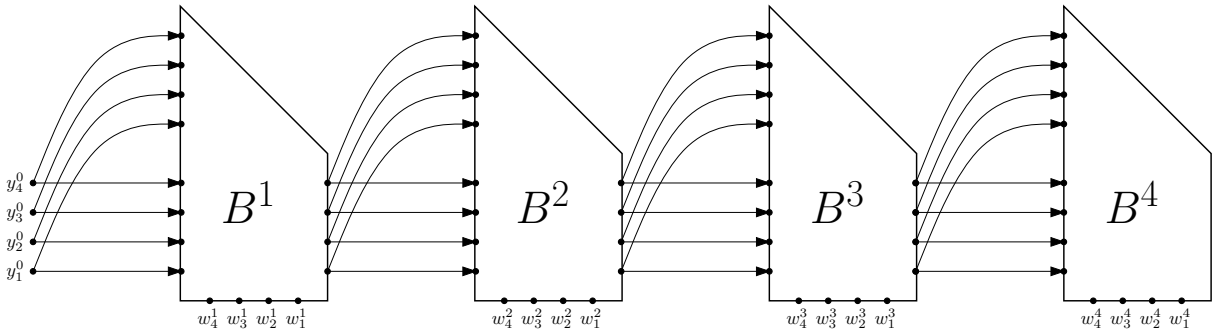

Figure 3: The UFP instance for the case $l = 4$, $k = 4$. All shown edges have capacity 2^l.

must be routed through a vertical path from $(k-j+1, k+j)$ to the sink. This immediately implies that any request $r_{(\hat{i},\hat{j})}$ with $\hat{i} > i$ cannot be routed since it must exit B^i through one of the y nodes and thus cross the path of request $r_{(i,j)}$ in one of the A junctions. In addition, a request $r_{(i,\hat{j})}$ with $\hat{j} < j$ cannot be routed since the sink w_j^i is located to the right of the vertical path and thus must cross it as well. By using the two observations above we conclude that at most one request can be routed through the extended UFP instance and its profit is 1.

The ratio $\frac{d_{max}}{u_{min}}$ created is at most 2^l and hence we choose $l = \lfloor \log(\frac{d_{max}}{u_{min}}) \rfloor$. In addition, we choose $k = |A|^{1/\epsilon}$. The number of edges used is $m = l(3k^2 + 4k + (k^2 + \frac{k(k-1)}{2}) \cdot |A|) = O(l \cdot k^2 \cdot |A|) = O(l \cdot k^{2+\epsilon})$. Therefore, the gap created is $lk = \Omega(m^{\frac{1}{2+\epsilon}} \sqrt{\lfloor \log(\frac{d_{max}}{u_{min}}) \rfloor})$ and by choosing a small ϵ the proof is completed.

■

6 Online Applications

6.1 Online Algorithms

Somewhat surprisingly, variants of the algorithms considered so far can be used in the online setting with slightly worse bounds. For simplicity, we present here an algorithm for the unweighted K-bounded UFP in which $r_j = d_j$ for every $j \in T$.

First note that for the unweighted K-bounded UFP, both $EKROUTE$ and $BKROUTE$ can be used as online deterministic algorithms since sorting the requests becomes unnecessary. By splitting T into T_1 and T_2 as in $CKROUTE$ we can combine the two algorithms:

$ONLINECKROUTE(T)$:
  Choose one of the two routing methods below with equal probabilities:
    Route T_1 by using $BKROUTE$ and reject all requests in T_2
    Route T_2 by using $EKROUTE$ and reject all requests in T_1

Theorem 6.1 *Algorithm $ONLINECKROUTE$ is an $O(K \cdot D^{\frac{1}{K}})$ competitive online algorithm for the unweighted K-bounded* UFP.

Proof: The expected value of the total accepted demand of the algorithm for any given input is the average between the total accepted demands given by the two routing methods. Since each method is $O(K \cdot D^{\frac{1}{K}})$ competitive on its part of the input, the theorem follows. ■

6.2 Online Lower Bound

In this section we show an $\Omega(K \cdot n^{\frac{1}{K}})$ lower bound for deterministic online algorithms in the unweighted K-bounded UFP. This matches the upper bound of $EKROUTE$ and slightly improves the previously known lower bound of $\Omega(n^{\frac{1}{K}})$ [1]. The lower bound is proved over a line network of length n. For simplicity, we assume that $n = r^K$ for some integer r. Otherwise, we can just use the largest r such that $r^K \leq n$ and prove the lower bound over a part of the line. All the requests are of demand $\frac{1}{K}$ and the edge capacities are all 1.

The lower bound can be represented by a subtree of a tree of height K with an outdegree of $n^{\frac{1}{K}}$. Each node in the subtree corresponds to one or more requests over some interval. The root corresponds to requests over the interval $[0, n]$. Node j in level $0 \leq i \leq K$ corresponds to the interval $[r^{K-i} \cdot j, r^{K-i} \cdot (j+1)]$. Note that the segments corresponding to a node's children are a partition of its own segment.

The lower bound is constructed together with its corresponding subtree by a DFS traversal of the tree. The traversal begins at the tree's root. At each node, the algorithm is given at most K requests over the interval corresponding to the current node. If the algorithm does not accept any of the K requests, the node's children are not traversed and the next node in the DFS traversal is visited. Otherwise, once the algorithm accepts a request, we start traversing each of its children recursively.

Note that in case we arrive at a leaf in the tree, the algorithm cannot accept any requests over its interval. That is because its interval is contained in K other accepted intervals; one for each of the node's ancestors. Therefore, the sequence of requests is well defined.

We need the following simple lemma for trees:

Lemma 6.2 *For a tree in which the out-degree of each node is either zero (a leaf) or $\delta > 1$, the number of leaves is at least $\delta - 1$ times larger than the number of internal nodes.*

Proof: We use induction on the tree growing from the root up. A tree with just one node (leaf) has the required property. Then, a tree that grows is a replacement of a leaf in an internal node and δ leaves. Therefore, the number of leaves has grown by $\delta - 1$ while the number of internal nodes has grown by 1. The required ratio is maintained. ■

Theorem 6.3 *The competitive ratio of any deterministic online algorithm for the K-bounded* UFP *is at least $\Omega(K \cdot n^{\frac{1}{K}})$.*

Proof: The algorithm's value from the input described above is the number of internal nodes in the subtree. That is because the algorithm accepts one request from each internal node but no requests from the leaves. A better solution for the same input is to accept all the requests represented by the leaves of the subtree. In that case, the value is K times the number of leaves since each leaf corresponds to K intervals. That is an allowed assignment since the intervals corresponding to the leaves do not intersect. By comparing the two solutions and using the previous lemma, we obtain the stated lower bound on the competitive ratio. ■

7 Conclusion

We presented algorithms based on combinatorial methods for all three variants of the UFP that either match or improve the previously known results. Due to their relatively simple description we believe that further analysis should lead to additional performance guarantees. Also, the algorithms might perform better over specific networks. An interesting open question is to find more

cases where combinatorial algorithms can replace or even improve algorithms based on randomized rounding.

Acknowledgments

We thank Amit Chakrabarti, Chandra Chekuri, Aravind Srinivasan, and Éva Tardos for helpful comments and useful pointers to the literature.

References

[1] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *34th IEEE Symposium on Foundations of Computer Science*, pages 32–40, 1993.

[2] A. Baveja and A. Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research*, 25(2):255–280, 2000.

[3] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[4] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. In *Proc. 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 51–66, 2002.

[5] C. Chekuri and S. Khanna. Edge disjoint paths revisited. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms*, pages 628–637, 2003.

[6] S. Fortune, J. Hopcroft, and J. Wyllie. The directed homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[7] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *Proc. 31st ACM Symp. on Theory of Computing*, pages 19–28, 1999.

[8] M.T. Hajiaghayi and F.T. Leighton. On the max-flow min-cut ratio for directed multicommodity flows. MIT technical report LCS-TR-910.

[9] R.M. Karp. *Reducibility among Combinatorial Problems, R.E. Miller and J.W. Thatcher (eds.), Complexity of Computer Computations*. Plenum Press, 1972.

[10] J. Kleinberg. *Approximation Algorithms for Disjoint Paths Problems*. PhD thesis, Massachusetts Institue of Technology, 1996.

[11] J. Kleinberg and É. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. In *Proc. 27th ACM Symp. on Theory of Computing*, pages 26–35, 1995.

[12] S. Kolliopoulos and C. Stein. Approximating disjoint-path problems using greedy algorithms and packing integer programs. In *Proc. 6th Integer Programming and Combinatorial Optimization Conference (IPCO)*, 1998.

[13] P. Kolman. A note on the greedy algorithm for the unsplittable flow problem. *Information Processing Letters*. To appear.

[14] P. Kolman and C. Scheideler. Simple on-line algorithms for the maximum disjoint paths problem. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 2001. To appear in *Algorithmica*.

[15] P. Kolman and C. Scheideler. Improved bounds for the unsplittable flow problem. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 628–637, 2002.

[16] B. Ma and L. Wang. On the inapproximability of disjoint paths and minimum Steiner forest with bandwidth constraints. *J. Comput. System Sci.*, 60(1):1–12, 2000.

[17] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[18] N. Robertson and P. D. Seymour. An outline of a disjoint paths algorithm. In *Paths, Flows and VLSI Design, Algorithms and Combinatorics*, volume 9, pages 267–292, 1990.

[19] N. Robertson and P. D. Seymour. Graph minors. XIII. the disjoint paths problem. *JCTB: Journal of Combinatorial Theory, Series B*, 63, 1995.

[20] A. Srinivasan. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In *Proc. 38th IEEE Symp. on Found. of Comp. Science*, pages 416–425, 1997.

[21] K. Varadarajan and G. Venkataraman. Graph decomposition and the greedy algorithm for edge-disjoint paths. In *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms*, 2004.