Strongly Polynomial Algorithms for the Unsplittable Flow Problem

Yossi Azar[1] and Oded Regev[2]

[1] Dept. of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel.
azar@math.tau.ac.il ***
[2] Dept. of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel.
odedr@math.tau.ac.il

Abstract. We provide the first strongly polynomial algorithms with the best approximation ratio for all three variants of the unsplittable flow problem (UFP). In this problem 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 as to maximize the total profit of the satisfied terminal pairs subject to the capacity constraints. Classical UFP, in which demands must be lower than edge capacities, is known to have an $O(\sqrt{m})$ approximation algorithm. We provide the same result with a strongly polynomial combinatorial algorithm. The extended UFP case is when some demands might be higher than edge capacities. For that case we both improve the current best approximation ratio and use strongly polynomial algorithms. We also use a lower bound to show that the extended case is provably harder than the classical case. The last variant is the bounded UFP where demands are at most $\frac{1}{K}$ of the minimum edge capacity. Using strongly polynomial algorithms here as well, we improve the currently best known algorithms. Specifically, for $K = 2$ our results are better than the lower bound for classical UFP thereby separating the two problems.

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 a 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 path, denoted EDP, is the special case where all demands, profits and capacities are equal to 1 (see [5]).

The EDP (and hence the UFP) is one of Karp's original NP-complete problems [6]. An $O(\sqrt{m})$ approximation algorithm is known for EDP [7] (for additional positive results see [12, 13]). Most of the results for UFP deal with the

*** Research supported in part by the Israel Science Foundation and by the US-Israel Binational Science Foundation (BSF).

classical case where $d_{max} \leq u_{min}$ (the maximal demand is at most the minimal capacity). The most popular approach seems to be LP rounding [2, 10, 14] with the best approximation ratio being $O(\sqrt{m})$ [2]. A matching lower bound of $\Omega(m^{1/2-\epsilon})$ for any $\epsilon > 0$ is shown in [5] for directed graphs. Both before and after the $O(\sqrt{m})$ result, there were attempts to achieve the same approximation ratio using combinatorial methods. Up to now however, these were found only for restricted versions of the problem [5, 10] and were not optimal. Our combinatorial algorithm not only achieves the $O(\sqrt{m})$ result for classical UFP but is also the first strongly polynomial algorithm for that problem.

The extended UFP is the case where both demands and capacities are arbitrary (specifically, some demands might be higher than some capacities). Due to its complexity, not many results addressed it. The first to attack the problem is a recent attempt by Guruswami et al. [5]. We improve the best approximation ratio through a strongly polynomial algorithm. By proving a lower bound for the extended UFP over directed graphs we infer that this case is really harder than the classical UFP. Specifically, for large demands we show that unless $P = NP$ it is impossible to approximate extended UFP better than $O(m^{1-\epsilon})$ for any $\epsilon > 0$.

Another interesting case is the bounded UFP case where $d_{max} \leq \frac{1}{K} u_{min}$ (denoted K-bounded UFP). It is a special case of classical UFP but better approximation ratios can be achieved. As a special case, it contains the half-disjoint paths problem where all the demands and profits are equal to $\frac{1}{2}$ and edge capacities are all 1 [8]. For $K \geq \log n$, a constant approximation is shown in [11] by using randomized rounding. For $K < \log n$, previous algorithms achieved an approximation ratio of $O(Kn^{\frac{1}{K-1}})$ ([2, 14] by using LP rounding and [3, 9] based on [1]). We improve the result to a strongly polynomial $O(Kn^{\frac{1}{K}})$ approximation algorithm which, as a special case, is a $O(\sqrt{n})$ approximation algorithm for the half disjoint case. Since this ratio is better than the lower bound for classical UFP, we achieve a separation between classical UFP and bounded UFP. The improvement is achieved by splitting the requests into a low demand set and a high demand set. The sets are treated separately by algorithms similar to those of [1] where in the case of high demands the algorithm has to be slightly modified. We would like to note that in our approximation ratios involving n, we can replace n with D where D is an upper bound on the longest path ever used (which is obviously at most n).

As a by-product of our methods, we provide online algorithms for UFP. Here, the network is known but requests arrive one by one and a decision has to be made without knowing which requests follow. We show on-line algorithms whose competitive ratio is somewhat worse than that of the off-line algorithms. We also show that one of our algorithms is optimal in the on-line setting by slightly improving a lower bound of [1].

We conclude this introduction with a short summary of the main results in this paper. We denote by d_{max} the maximum demand and by u_{min} the minimum edge capacity.

- Classical UFP ($d_{max} \le u_{min}$) - Strongly polynomial $O(\sqrt{m})$ approximation algorithm.
- Extended UFP (arbitrary d_{max}, u_{min}) - 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 ($d_{max} \le \frac{1}{K}u_{min}$) - Strongly polynomial $O(Kn^{\frac{1}{K}})$ approximation algorithm.

2 Notation

Let $G=(V,E)$, $|V|=n$, $|E|=m$, be a (possibly directed) graph and a capacity function $u: E \to \mathbf{R}^+$. 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 $j \in T = \{1, ..., l\}$. 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 $d_{max} \le u_{min}$ assumption made in classical UFP.

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

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

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

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,

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

3 Algorithms for UFP

3.1 Algorithm for Classical UFP

In this section we show a simple algorithm for classical UFP (the case in which $d_{max} \leq u_{min}$). The algorithm's approximation ratio is the same as the best currently known algorithm. Later, we show that unlike previous algorithms, this algorithm can be easily made strongly polynomial and that it can even be used in the extended case.

We split 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_{min}, 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_{max}, 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

$Routine_2(S)$:
  foreach k from $\lfloor \log \alpha_{min} \rfloor$ to $\lceil \log \alpha_{max} \rceil$
    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
  foreach $j \in S$ 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 <u>route</u> the request on P and for $e \in P$ set $L_j(e) = L_{j-1}(e) + \frac{d_j}{u(e)}$
    else <u>reject</u> the request

Theorem 1. *Algorithm $PROUTE$ is an $O(\sqrt{m})$ approximation algorithm for classical UFP.*

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

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

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 $r(\{j \in \mathcal{Q}' | F(j, Q_j) > \alpha'\}) \geq r(\mathcal{Q})/4$. 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 $\mathcal{Q}'_{high} = \{j \in \mathcal{Q}' | F(j, Q_j) > \alpha'\}$ and $\mathcal{Q}'_{low} = \{j \in \mathcal{Q}' | F(j, Q_j) \leq 2\alpha'\}$ 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 $E_{heavy} = \{e \in E | L_l(e) \geq \frac{1}{4}\}$ 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,

$$r(\mathcal{P}) = \sum_{j \in \mathcal{P}} F(j, P_j) \sum_{e \in P_j} \frac{d_j}{u(e)}$$

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

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 $R = Q'_{high} \setminus P$. We compare the profit given by our algorithm to that found in R by using Lemma 1. Since $\frac{r_j}{d_j}$ is a non increasing sequence, it is enough to bound the total demand routed in prefixes of the two sets. For that we use the notation $R^k = R \cap \{1, ..., k\}$ and $\mathcal{P}^k = \mathcal{P} \cap \{1, ..., k\}$ for $k = 1, ..., l$. For each request $j \in R^k$ 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 $E^k = \{e_j | j \in R^k\}$.

Lemma 2. $E^k \subseteq E_{heavy}$

Proof. Let $e_j \in E^k$ be an edge with $j \in R^k$, a request corresponding to it. We claim that when the algorithm fails finding 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 R^k is routed through an edge of E^k in the optimal solution, $d(R^k) \leq \sum_{e \in E^k} u(e)$. 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 2, $|E^k| \leq |E_{heavy}| < \sqrt{m}$ and hence,

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

We use Lemma 1 by combining the inequality above on the ratio of demands and the nonincreasing sequence $\frac{r_j}{d_j}$. This yields

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

or,

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

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

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

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 $\log \frac{\alpha_{max}}{\alpha_{min}}$ times. Therefore, it is polynomial but still not strongly polynomial. We add a preprocessing step whose purpose is to bound the ratio $\frac{\alpha_{max}}{\alpha_{min}}$. 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 $S_{tiny} = \{j \in S | d_j \leq \frac{u_{min}}{l}\}$ on any simple path.
> Run *Routine*$_2(S \setminus S_{tiny})$.

Theorem 2. *Algorithm SPROUTE is a strongly polynomial $O(\sqrt{m})$ approximation algorithm for 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 $k \cdot d_{max}$. 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 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:

$ESPROUTE(T)$: for any $1 \leq i \leq 2 + \max\{\lceil \log d_{max}/u_{min} \rceil, 0\}$ such that T_i is not empty   run $Routine_3(T_i)$ on the resulting graph choose the best solution obtained

The proof of the following theorem is left to Appendix A.1:

Theorem 3. *Algorithm $ESPROUTE$ is a strongly polynomial $O(\sqrt{m}\log(2 + \frac{d_{max}}{u_{min}}))$ approximation algorithm for extended UFP.*

4 Algorithms for K-bounded UFP

In the previous section we considered the classical UFP in which $d_{max} \leq u_{min}$. We also extended the discussion to extended UFP. In this section we show better algorithms for K-bounded UFP in which $d_{max} \leq \frac{1}{K}u_{min}$ where $K \geq 2$.

4.1 Algorithms for Bounded Demands

In this section we present two algorithm for 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.

$EKROUTE(T)$:
$\mu \leftarrow 2D$
sort the requests in T according to a non-increasing order of r_j/d_j
foreach $j \in T$ 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
foreach $j \in T_i$ 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 $BKROUTE$ in instances where the demands are in the correct range.

The proof of the following theorem can be found in Appendix A.2:

Theorem 4. *Algorithm $EKROUTE$ is a strongly polynomial $O(K \cdot D^{\frac{1}{K}})$ approximation algorithm for 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 K-bounded UFP.*

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

Proof. Let 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 $\frac{r(Q)}{2}$ 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 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 6. *[4] 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 7. *For any $\epsilon > 0$, 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 extended UFP composed of l copies of A, $A^1, A^2, ..., A^l$ 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. The above structure is shown in the following figure for the hypothetical case where $l = 4$. 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.

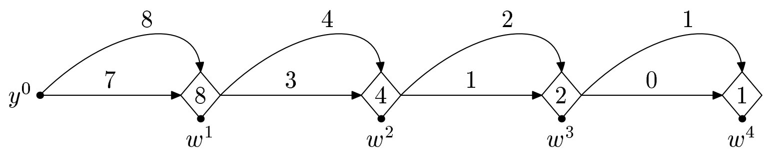

Fig. 1. The UFP instance for the case $l = 4$

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

$[y^0, x^1, y^1, x^2, y^2, ..., y^{i-1}, z^i, w^i]$. 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 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 8. *For any $\epsilon > 0$ 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. Omitted.

6 Online Applications

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 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 9. *Algorithm $ONLINECKROUTE$ is an $O(K \cdot D^{\frac{1}{K}})$ competitive online algorithm for 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.

Theorem 10. *The competitive ratio of any deterministic on-line algorithm for the K-bounded UFP is at least $\Omega(K \cdot n^{\frac{1}{K}})$.*

Proof. Omitted.

7 Conclusion

Using combinatorial methods we showed algorithms for all three variants of the UFP problem. We improve previous results and provide the best approximations for UFP by using strongly polynomial algorithms. Due to their relative simplicity we believe that further analysis should lead to additional performance guarantees such as non linear bounds. Also, the algorithms might perform better over specific networks. It is interesting to note that no known lower bound exists for the half-disjoint case and we leave that as an open question.

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. *To appear in Mathematics ofOperations Research.*

[3] A. Borodin and R. El-Yaniv. Online computation and competitive analysis (cambridge university press, 1998). *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 29, 1998.

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

[5] 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. *Proc. of STOC '99*, pages 19–28.

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

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

[8] J. Kleinberg. Decision algorithms for unsplittable flow and the half-disjoint paths problem. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC '98)*, pages 530–539, New York, May 23–26 1998. ACM Press.

[9] J. Kleinberg and É. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. *Proc. of STOC '95*, pages 26–35.

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

[11] P. Raghavan and C.D. Thompson. Provably good routing in graphs: Regular arrays. In *Proc. 17th ACM Symp. on Theory of Computing*, May 1985.

[12] 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.

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

[14] 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.

A Appendix

A.1 Proof of Theorem 3

Proof. The proofs of Theorem 1 and of Theorem 2 hold also for the extended case. The only part which has to be proved is Lemma 2. The following replaces the lemma:

Lemma 3. $E^k \subseteq E_{heavy}$

Proof. Let $e_j \in E^k$ be an edge with $j \in R^k$, a request corresponding to it. We claim that when the algorithm fails finding 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 $2^{i'-3}u_{min} \geq u(e_j)/4$.

The number of iterations $ESPROUTE$ performs is at most l since we ignore empty T_i's. 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.

A.2 Proof of Theorem 4

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 algorithm. 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 $\mathcal{Q}^k = \mathcal{Q} \cap \{1, ..., k\}$ and $\mathcal{P}^k = \mathcal{P} \cap \{1, ..., k\}$ for $k = 1, ..., l$. 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 $R^k = Q^k \setminus P^k$. For $j \in R^k$, 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,

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

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

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

By combining the two inequalities shown above,

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

The algorithm followed a non-increasing order of $\frac{r_j}{d_j}$ and by Lemma 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.