A Quasi-PTAS for Unsplittable Flow on Line Graphs

Nikhil Bansal
IBM T. J. Watson Research Center
Yorktown Heights, NY, USA
nikhil@us.ibm.com

Amit Chakrabarti∗
Department of Computer Science
Dartmouth College
Hanover, NH, USA
ac@cs.dartmouth.edu

Amir Epstein†
School of Computer Science
Tel Aviv University
Tel Aviv, Israel
amirep@tau.ac.il

Baruch Schieber
IBM T. J. Watson Research Center
Yorktown Heights, NY, USA
sbar@us.ibm.com

ABSTRACT

We study the Unsplittable Flow Problem (UFP) on line graphs and cycles, focusing on the long-standing open question of whether the problem is APX-hard. We describe a deterministic quasi-polynomial time approximation scheme for UFP on line graphs, thereby ruling out an APX-hardness result, unless NP ⊆ DTIME(2polylog(n)). Our result requires a quasi-polynomial bound on all edge capacities and demands in the input instance. We extend this result to undirected cycle graphs.

Earlier results on this problem included a polynomial time (2 + ε)-approximation under the assumption that no demand exceeds any edge capacity (the "no-bottleneck assumption") and a super-constant integrality gap if this assumption did not hold. Unlike most earlier work on UFP, our results do not require a no-bottleneck assumption.

Categories and Subject Descriptors

F.2.2 [Analysis of Algorithms and Problem Complexity]: Non-numerical Algorithms and Problems

General Terms

Algorithms, Theory

Keywords

unsplittable flow, resource allocation, scheduling, approximation algorithms, approximation scheme

∗Supported by an NSF CAREER award and startup funds from Dartmouth College.
†Supported in part by the German-Israeli Foundation and by the Israel Science Foundation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2006 ACM 1-59593-134-1/06/0005 ...$5.00.

1. INTRODUCTION

1.1 Problem Definition and Motivation

The unsplittable flow problem (UFP) asks for the maximum profit subset of a given set of point-to-point flow demands that can be simultaneously routed in a given capacitated network. To be precise, the input consists of a graph G = (V, E) with edge capacities {c_e} e∈E that represent the amount of a fungible resource available on each edge, and a set of n 4-tuples \{(s_i, t_i, \rho_i, w_i)\}_{i=1}^n that represent the flow demands: the \(s_i\)th demand is to be routed from a source \(s_i \in V\) to a sink \(t_i \in V\), requires \(\rho_i\) units of the resource and yields a profit in the amount \(w_i\) if it is routed. Each \(c_e\), \(\rho_i\) and \(w_i\) is an integer in the range \([1, L]\), for some (large) integer \(L\).

A subset \(S \subseteq \{1, \ldots, n\}\) of the demands is said to be feasible if all demands in \(S\) can be simultaneously routed, with each demand using one path in \(G\) (hence, “unsplittable”) and respecting the capacity constraints on every edge in \(G\). The goal is to compute a feasible \(S\) that maximizes the profit \(w(S) := \sum_{s \in S} w_s\).

In this paper, we study UFP on line and cycle networks. For line networks, the graph \(G\) is an undirected line graph with \(m\) edges. In this case we can orient the line arbitrarily to form a directed "left-to-right" path of length \(m\) and identify \(V\) with \([0, 1, \ldots, m]\) in the natural left-to-right order along this path. Note also that choosing a path for each routed demand is a non-issue since there is no choice in a line network (indeed, in any tree network). For cycle networks, the graph \(G\) is an undirected cycle graph with \(m\) edges. In this case we need to determine for each routed demand which one of the two possible routes it uses.

The study of UFP is motivated by its being a common generalization of a number of combinatorial optimization problems with a wide variety of applications. Even with the underlying graph restricted to a line graph, UFP continues to be fairly general problem. For instance, if the graph is a single edge, it simplifies to the Knapsack problem, and if all resource requirements and edge capacities are set to unity, it simplifies to the maximum weight independent set problem in an interval graph.

Applications of UFP on line graphs typically view the graph as

1The upper bound on the \(w_i\)'s is not really necessary when designing an approximation algorithm; see Section 2.
a discretization of a timeline and the terminals $s_i$ and $t_i$ as start and finish times of a set of jobs. Under this view, UFP becomes the problem of scheduling jobs with known machine requirements given the number of machines available at each time unit. In fact, an IBM data center recently dealt with an instance of this problem, with each edge (time unit) viewed as a shift and the capacity of an edge viewed as the number of people available at that shift. The problem was to schedule long activities spanning over several shifts. Other areas where the problem has come up before include bandwidth allocation of sessions in communication networks [5], the general caching problem with varying page size [1, 5] and the scheduling of projects for space missions [14]. Accordingly, the problem has come to be known by a number of names such as “Resource Allocation,” “Bandwidth Allocation,” “Resource Constrained Scheduling” and “Call Admission Control.”

### 1.2 Background and Prior Work

The unsplittable flow problem is clearly NP-complete, because when the graph $G$ is a single edge with all demands going across it, UFP simply becomes the Knapsack problem. Two other noteworthy special cases of UFP are the Edge Disjoint Paths Problem (EDPP), obtained by setting each edge capacity and resource requirement to 1, and Uniform Capacity UFP (UCUFP), obtained by equalizing all edge capacities. EDPP has long been known to be NP-complete on directed graphs with 2 terminal pairs [11] and on undirected graphs with a non-constant number of terminal pairs [15]. It is also known to have an $O(\sqrt{|E|})$-approximation algorithm on directed graphs $G = (V, E)$ [16]. This result was generalized to UCUFP by Srivinasa [19] and eventually to UFP by Baveja and Srivinasa [6]. The result for UFP required the assumption $p_{\text{max}} \leq c_{\text{min}}$, which has come to be known as the no-bottleneck assumption. Azar and Regev [4] provided a combinatorial and easy-to-analyze algorithm achieving the same $O(\sqrt{|E|})$ approximation ratio for UFP, under the no-bottleneck assumption. Chekuri and Khanna [9] refined the approximation ratio for EDPP to $O\left(\min\{n^{2/3}, |E|^{1/2}\}\right)$.

On the lower bound side, Guruswami et al. [13] showed that EDPP is hard to approximate on directed graphs to a factor in $\Omega(|E|^{1/2-\varepsilon})$ unless $P = NP$. Recently, Andrews and Zhang [3] gave the first nontrivial hardness result for EDPP on undirected graphs, and in follow-up work Andrews et al. [2] prove the current best bound of $\Omega(\log^{1/2+\varepsilon}(|E|))$ under the assumption $NP \not\subseteq ZPTIME(2^{\text{polylog}(n)})$. Clearly, these hardness results also hold for UFP with the no-bottleneck assumption as EDPP is a special case of this problem. Without the no-bottleneck assumption, Azar and Regev [4] showed that UFP is hard to approximate within a factor of $\Omega(|E|^{1/3})$ unless $P = NP$. UFP is also known to be APX-hard for very simple classes for graphs. In particular, Garg, Vazirani and Yannakakis [12] implicitly showed that UFP is APX-hard for the case of depth-3 trees, even when all demands are 1, and all edge capacities are either 1 or 2.

Turning to line graphs — our primary focus in this work — we note that UCUFP on line graphs has been studied for quite a while, under the various resource allocation and scheduling guises mentioned earlier. The first constant factor approximation for UCUFP on a line was given by Phillips, Uma and Wein [17] who obtained a 6-approximation. This factor was then improved by Bar-Noy et al. [5] to 3 and by Calinescu et al. [7] to $(2 + \varepsilon)$. For UFP on line graphs, the first constant factor approximation was due to Chakrabarti, Chekuri, Gupta and Kumar [8]; subsequent work of Chekuri, Mydlarz and Shepherd [10] provided a $(2+\varepsilon)$-approximation. Both results require the no-bottleneck assumption. The former paper proves an integrality gap of $\Omega(\log(p_{\text{max}}/p_{\text{min}}))$ for the natural LP relaxation when this assumption is removed.

Finally, turning to UFP on cycle graphs — which we also focus on — we note that both Chakrabarti et al. [8] and Chekuri et al. [10] gave approximation results comparable to their respective results on line graphs. In earlier work, Schrijver, Seymour and Winkler [18] studied a congestion minimization version of the problem and noted that the problem is applicable to the planning of certain optical communication networks.

### 1.3 Our Work

The main result of our work is the following theorem.

**Main Theorem.** There is a quasi-polynomial time approximation scheme for UFP on line and cycle graphs, provided all capacities and resource requirements are integers bounded by $2^{\text{polylog}(n)}$.

Importantly, in contrast to previous work, we do not require a no-bottleneck assumption. Of course, in view of the integrality gap result of Chakrabarti et al. [8], to achieve our performance bound we were unable to analyze our algorithm’s solution quality by comparing it to the LP optimum.

Given that UFP is APX-hard even for very simple classes of graphs, the question of whether one can prove APX-hardness for UFP on line graphs (and perhaps even for the simpler Resource Allocation Problem, i.e., UCUFP on a line) had been a longstanding open question. Our work partially resolves this question and suggests that a PTAS is likely for this problem.

**Our Techniques and Comparison with Earlier Work:** There is a common thread that runs through the earlier $(2+\varepsilon)$-approximation for UCUFP on a line [7] and through both earlier papers achieving $O(1)$-approximation for UFP on a line [8, 10]. All of these algorithms start by partitioning the demands in the input instance into “large” and “small” ones and solve the two resulting subproblems separately. In an instance with all large demands, a feasible solution can only have limited overlap, so an exact optimum can be found via dynamic programming. In an instance with all small demands, a linear programming relaxation turns out to have low integrality gap and can be suitably rounded. The earlier algorithms proceed by picking either an all-large or an all-small solution, thereby necessarily losing a factor of 2. One important contribution of this work is that we show, for the first time, how to effectively combine “large” and “small” demands, thereby breaking the factor-2 barrier.

Our algorithm for a line uses a simple recursive approach. It considers the (combinatorial) mid-point of the line graph in the given UFP instance, “guesses” the demands that cross this mid-point in some optimal solution, and then recurses on both halves of the line. While an exact solution could require trying up to exponentially many guesses at each step, we show that it suffices (up to a factor $(1+\varepsilon)$) to choose the guess from a quasi-polynomially large collection $S$. In particular, for any arbitrary collection $D$ of demands crossing a point, there is another set $D'$ of demands in the collection $S$, that has almost identical profit and more crucially has the property that its resource requirements never exceed those of $D$ at any edge $e$ in the graph. We extend this algorithm to undirected cycle using techniques adapted from Schrijver, Seymour and Winkler [18] to determine how the demands are routed.

The rest of the paper is organized as follows. Sections 2 to 4 describe the algorithm for line graphs, Section 5 describes the extension to cycle graphs, and Section 6 gives some concluding remarks and open problems.
2. PRELIMINARIES

We can assume, w.l.o.g., that each source lies “to the left” of the corresponding sink, i.e., that \( s_i < t_i \) for each \( i \). By appropriate rescaling, we can assume that \( \rho_{\text{min}} \geq 1/L \). It is also possible to discard all demands with profit less than \((\delta/n)w_{\text{max}}\) incurring a loss of at most a \( \delta \) fraction of the optimal profit; doing so ensures \( w_{\text{max}} \leq n/\delta \). Therefore, we do not really need an upper bound on the \( w_i \)’s in the input instance.

For any \( i \in \{1, \ldots, n\} \), the ratio \( w_i/\rho_i \) is called the profit density of demand \( i \). Note that after the above adjustments we have \( 1 \leq w_i/\rho_i \leq L n/\delta \) for every demand \( i \). We partition the set of demands in the input instance into classes (several of which may be empty) based on their profit densities: class \( q \) consists of demands \( i \) with \( 2^q-1 \leq w_i/\rho_i < 2^q \). This gives a total of at most \( Q := 1 + \left\lfloor \log_{\max} \{w_i/\rho_i\} \right\rfloor \leq \log(n) \) classes, provided \( L \leq 2^{\log(\log(n))} \).

Any vertex that is neither a source nor a sink can be “removed” from the UFP instance, leaving it combinatorially unchanged. To be precise, if \( u \in V \) is such a vertex and \( 0 < u < m \), then we can delete \( u \) and merge the edges \( e = \{u-1, u\} \) and \( e' = \{u, u+1\} \) into a single edge with capacity \( \min\{c_e, c_{e'}\} \); if \( u = 0 \) or \( u = m \), we can simply delete \( u \) from the graph. After doing so repeatedly, we may assume that every vertex is a terminal (either a source or a sink), so that \( m \leq 2n \). Thus, the complexity of an algorithm for UFP on a line network can be described in terms of \( n \) and \( L \) alone, ignoring \( m \).

For any \( i \in \{1, \ldots, n\} \), the half-open interval \( [s_i, t_i] \) is called the interval of demand \( i \). We say that this demand spans an edge \( e = \{u-1, u\} \in E \) if \( (s_i, t_i) \supseteq u \), lies to the left of \( e \) if \( t_i \leq u-1 \) and lies to the right of \( e \) if \( s_i \geq u \). Let \( S \subseteq \{1, \ldots, n\} \) be a subset of the demands. The load of \( S \) on edge \( e \) is defined to be the total amount of resource used by those demands in \( S \) that span \( e \):

\[
\text{load}(S,e) := \sum_{i \in S: (s_i, t_i] \supseteq u} \rho_i, \quad \text{where } e = \{u-1, u\}.
\]

We define a profile to be an \( m \)-dimensional vector indexed by the edges of \( G \) and having nonnegative real entries. For example, the edge capacities \( \{c_e\}_{e \in E} \) in the given UFP instance form a profile; we denote this profile \( c \). Another natural example is the usage profile (or simply, profile) of a set \( S \) of demands, defined to be the vector of its loads on all the edges of \( G \):

\[
\text{prof}(S) := \left(\text{load}(S, \{0, 1\}), \text{load}(S, \{1, 2\}), \ldots, \text{load}(S, \{m-1, m\})\right).
\]

We use operators such as “\( \leq \)” and “\(+\)” on profiles in the standard coordinate-wise manner. Therefore, we can express the feasibility of \( S \) by writing \( \text{prof}(S) \leq c \). The set \( S \) is said to be a pile at edge \( e \) if every demand in \( S \) spans \( e \).

It is not hard to see that the profile of a pile is a unimodal sequence. When graphed, such a profile looks like a stepped mountain. Imagine imposing a coarse uniform grid of horizontal lines on this graph and requiring the horizontal segments in the graph to lie on grid lines. Doing so greatly restricts the profile and drastically reduces its description complexity. This observation is one of the key insights behind our algorithm and it motivates the following definition.

**Definition 2.1.** Let \( e = \{u-1, u\} \) be an edge of \( G \) and \( h \) and \( \delta \) be positive reals with \( h \leq c_e \), \( \delta \leq 1 \) and \( 1/\delta \) an integer. Let \( x_1, \ldots, x_{1/\delta} \) and \( y_1, \ldots, y_{1/\delta} \) be vertices of \( G \) with

\[
x_1 \leq x_2 \leq \cdots \leq x_{1/\delta} \leq u-1, \quad \text{and}
\]

\[
u \leq y_1 \leq \cdots \leq y_{1/\delta} \leq y_1.
\]

Then the profile \( (\ell_1, \ldots, \ell_m) \), where

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

is said to be a \( \delta \)-restricted profile with peak \( c \) and height \( h \), parameterized by the \( x \)'s and \( y \)'s. This particular profile is denoted \( \text{RFP}_c(e; h; x_1, \ldots, x_{1/\delta}; y_1, \ldots, y_{1/\delta}) \).

Note that by not requiring the \( x \)'s and \( y \)'s to be distinct, we allow the “step size” of such a restricted profile at one of these vertices to be greater than \( \delta h \); it can be a larger multiple of \( \delta h \).

It is clear from the definition that the number of \( \delta \)-restricted profiles in an \( m \)-edge line graph with a given peak and a given height is exactly the number of valid settings of the parameters \( x_1, \ldots, x_{1/\delta} \).
and $y_1, \ldots, y_{1/\delta}$. Using the very loose bound that each $x_i$ and each $y_j$ has at most $m$ valid settings, we obtain the following fact. It will be very useful in the sequel.

**Fact 2.2.** There are at most $m^{2/\delta}$ $\delta$-restricted profiles with a given peak and a given height. \hfill \Box

### 3. TWO LEMMAS ABOUT RESTRICTED PROFILES

Consider any optimal solution $S^* \subseteq \{1, \ldots, n\}$ to the given UFP instance and any edge $e \in E$. The subset of demands in $S^*$ that span $e$ form a pile; let $T^*$ be this subset. The two lemmas in this section will be used in the analysis of our UFP algorithm to show that it eventually computes a set $T$ of demands that yields a $(1 - O(\delta))$ fraction of the profit of $T^*$ and such that $\text{prof}(T)$ is dominated by a $\delta$-restricted profile $\pi$ that approximates and is in turn dominated by $\text{prof}(T^*)$, for some appropriate small $\delta$. The lemmas need some additional restrictions on the demands in $T^*$ that are made precise below. The first of these lemmas is an existence result for a suitable restricted profile $\pi$. The second lemma is an algorithmic result that shows that given a suitable $\pi$ one can compute a high-profit subset $T$ efficiently.

Throughout this section we shall assume that $0 < \delta < 1$ and that $1/\delta$ is an integer.

**Lemma 3.1.** Let $S$ be a pile at edge $e$ of class-$q$ demands and let $B$ be such that $\rho_i \leq B$ for all $i \in S$. Let $h$ be the largest integer multiple of $\rho_{\text{min}}$ that does not exceed $\text{load}(S, e)$. Then there exists a $\delta$-restricted profile $\pi$ with peak $e$ and height $h$ and a subset $T \subseteq S$ such that

1. $\text{prof}(T) \leq \pi \leq \text{prof}(S), \text{ and}$
2. $w(S \setminus T) \leq 2^{q+1}(\delta h + B)$.

**Proof.** The idea is to “scan” the edges of $G$ from left to right and mark off the first edge on which the load of $S$ is at least $\delta h$, at least $2\delta h$, etc.; these edges then define the left half of the profile $\pi$ and the right half of $\pi$ is defined similarly. An example of this construction is shown in Figure 1.

To be precise, define

$$x_j = \min\{i : \text{load}(S, \{i, i + 1\}) \geq jh\}, \text{ for } 1 \leq j \leq 1/\delta,$$

$$y_j = \max\{i : \text{load}(S, \{i - 1, i\}) \geq jh\}, \text{ for } 1 \leq j \leq 1/\delta.$$

and let $\pi := \text{RP}_e(c; h; x_1, \ldots, x_{1/\delta}; y_1, \ldots, y_{1/\delta})$. It follows from the construction that $\pi \leq \text{prof}(S)$.

It also follows that every entry of the vector $(\text{prof}(S) - \pi)$ is upper bounded by $\delta h$, except for the entries indexed by edges between vertices $x_{1/\delta}$ and $y_{1/\delta}$, which are upper bounded by $\rho_{\text{min}}$. Thus, to construct a subset $T \subseteq S$ such that $\text{prof}(T) \leq \pi$, we can apply the following greedy procedure. Order all demands in $S$ by their left end-points (i.e., $s_i$) from left to right and greedily select demands until the total resource requirement of the selected demands is at least $\delta h$; let $X$ be the set of demands thus selected. Then order all demands by their right end-points (i.e., $t_i$) from right to left and do the same; let $Y$ be the resulting set. Let $T := S \setminus (X \cup Y)$.

We now have a set $T$ that satisfies property 1. But note that the total resource requirement of the demands in $X$ is at most $\delta h + B$, because each individual demand $i \in S$ has $\rho_i \leq B$. The same is true for $Y$. Since all demands under consideration are in class $q$, the total profit in $X \cup Y$ is at most $2(\delta h + B) \cdot 2^q = 2^{q+1}(\delta h + B)$, which shows that $T$ satisfies property 2. \hfill \Box

We present a linear programming based algorithm (Algorithm 1: PILE-PACK) to pack a high-profit subset of a given set of demands into a given restricted profile. Our next lemma proves a basic property of this algorithm.

**Lemma 3.2.** Let $S$ be a pile at edge $e$ of class-$q$ demands such that $\rho_i \leq B$ for all $i \in S$, and let $\pi$ be a $\delta$-restricted profile with peak $e$. Then $\text{prof}(T^*) \leq \pi$ and let $T^*$ be the set returned by PILE-PACK($G, \pi, S$). Then $w(T^*) - w(T) \leq 2^{q+1}B/\delta$.

**Proof.** Assume, w.l.o.g., that no deletions are necessary in Step 1 of PILE-PACK. Note that if the linear program (2.1)-(2.3) were solved with the added condition $\alpha_i \in \{0, 1\}$ for $i \in S$, then the set returned by PILE-PACK would be a maximum profit subset of $S$ that fit the profile $\pi$. Therefore, if $W$ denotes the (fractional) optimum of the LP, we have $w(T^*) \leq W$.

A vertex solution of the LP must make $|S|$ inequalities tight. The total number of distinct inequalities given by (2.1) and (2.2) is

**Algorithm 1: PILE-PACK ($G, \pi, S$)**

**Input:** line graph $G = (V, E)$, profile $\pi = \text{RP}_e(c; h; x_1, \ldots, x_{1/\delta}; y_1, \ldots, y_{1/\delta})$, set $S \subseteq [n]$ indexing demands

**Output:** a subset $T \subseteq S$ with $\text{prof}(T) \leq \pi$

1. Delete from $S$ all demands $i$ where $s_i < x_1$ or $t_i > y_1$
2. for $j = 1$ to $1/\delta - 1$
3. $A_j \leftarrow \{i \in S : s_i < x_{j+1}\}$
4. $B_j \leftarrow \{i \in S : t_i > y_{j+1}\}$
5. $A_{j \delta} \leftarrow B_{j \delta} \leftarrow S$
6. Compute a vertex solution to the following linear program in the variables $\{\alpha_i\}_{i \in S}$:

   maximize $\sum_{i \in S} w_i \alpha_i$, for nonnegative $\alpha_i$, s.t.

   $\sum_{i \in A_j} \rho_i \alpha_i \leq j\delta h$, for $1 \leq j \leq 1/\delta$ (2.1)

   $\sum_{i \in B_j} \rho_i \alpha_i \leq j\delta h$, for $1 \leq j \leq 1/\delta$ (2.2)

   $\alpha_i \leq 1$, for $i \in S$ (2.3)

7. $T \leftarrow \{i \in S : \alpha_i = 1\}$
8. return $T$
Algorithm 2: Line-UFP-Recursive (G, c, S)

Input: line graph G = (V, E), profile c of edge capacities \{c_e : e ∈ E\}, set S ⊆ [n] indexing demands \{(s_i, t_i, w_i, ρ_i) : 1 ≤ i ≤ n\}
Output: a subset of S that is feasible w.r.t. c and has profit ≥ (1 − O(δ)) · OPT

1. if |S| ≤ 1 then
2. if prof(S) ≤ c then return \emptyset
3. find an edge e∗ ∈ E such that |LEFT(e∗)| ≤ n/2 and |RIGHT(e∗)| ≤ n/2
4. for q = 1 to Q do
   S_q ← \{i ∈ span(e∗) : 2^{q−1} ≤ w_i/ρ_i < 2^q\}
5. foreach Q-tuple (T_1, . . . , T_Q) with each T_q ⊆ S_q and |T_q| ≤ 1/δ^2 do
6. \[ T ← \bigcup_{q=1}^Q T_q \]
7. if all demands in T can be routed then
8. route all demands in T and obtain residual capacities \{c_e′ : e ∈ E\}
9. foreach (h_1, . . . , h_Q) ∈ R^q with each h_q an integer multiple of ρ_min and h_1 + · · · + h_Q ≤ c_e′ do
10. for q = 1 to Q do
   S_q,small ← \{i ∈ S_q \setminus T_q : ρ_i ≤ \delta^q(h_q + ρ_min + load(T_q, e∗))\}
11. foreach Q-tuple (π_1, . . . , π_Q) with each π_q a δ-restricted profile with peak e∗ and height h_q, such that π_1 + · · · + π_Q ≤ c′ do
12. for q = 1 to Q do
   U_q ← PILE-PACK(G, π_q, S_q,small)
13. U ← \bigcup_{q=1}^Q U_q
14. route all demands in U and obtain residual capacities \{c_e′′ : e ∈ E\}
15. L ← LINE-UFP-RECURSIVE(G_{L,e∗,c′′,LEFT(e∗)}
16. R ← LINE-UFP-RECURSIVE(G_{R,e∗,c′′,RIGHT(e∗)}
17. record the solution T ∪ U ∩ L ∪ R
18. return the most profitable of the recorded solutions

at most 2/δ. Therefore, at most 2/δ of the |S| variables \{α_s\} can be fractional and cause the corresponding demands to be discarded from the fractional solution when forming the set T. Each of these discarded demands has resource requirement at most B, and thus (since it is a class-q demand) profit at most 2^{q}B. Therefore W − w(T) ≤ (2/δ) · 2^{q}B = 2^{q+1}B/δ and the lemma follows. □

4. THE FINAL ALGORITHM

We are ready to describe our algorithm for UFP on a line graph. We begin with some notation and give an intuitive outline of the algorithm.

For each edge e = \{u − 1, u\} of the graph G, let G_{L,e} denote the subgraph of G induced by the vertices \{0, 1, . . . , u\} and let G_{R,e} denote the subgraph of G induced by the vertices \{u, u + 1, . . . , \}. Let LEFT(e), RIGHT(e) and span(e) denote the subset of demands in the input that (respectively) lie to the left of e, lie to the right of e and span e.

Recall that the demands in the input are partitioned into Q ≤ polylog(n) classes based on their profit densities. Consider the set of class-q demands that span e in an optimal solution. Some of these demands are “large” (have a high resource requirement) whereas the rest are “small.” The large demands cannot be too many in number, so the algorithm can try out all possible subsets till it hits the “right” one. As for the small demands, they form a pile at e and, by Lemma 3.1, the profile of such a pile can be approximated by an appropriate δ-restricted profile with peak e, with the smallness of the demands ensuring that the profit reduction from such an approximation is tiny. Our algorithm tries out all possible settings of the parameters of such a profile. For each choice, it then uses the algorithm behind Lemma 3.2 to pack a high-profit set of demands into the profile. Finally, it recursively solves the resulting subproblems on the graphs G_{L,e} and G_{R,e}; note that these two subproblems are completely decoupled as we have already handled the demands that span e. Moreover since the δ-restricted profile is dominated by the pile of demands that span e in the optimal solution, this ensures that the capacity available to the algorithm in the recursive step is no less than that in the optimal solution, and the recursion can be applied meaningfully. Fact 2.2 upper bounds the amount of guesswork done by the algorithm, and thus, its running time.

A formal description of the algorithm is shown above in Algorithm 2: Line-UFP-Recursive.

The following two theorems establish the key properties of the algorithm.

**Theorem 4.1.** Algorithm LINE-UFP-RECURSIVE runs in time quasi-polynomial in n, provided L is quasi-polynomial in n.

**Proof.** We upper bound the number of iterations of each of the three nested loops. There are at most n^{1/δ^2} possibilities for each set T_q, so the loop beginning at line 5 runs for at most n^{Q/δ^2} iterations. Since \( c_{\text{max}}/\rho_{\text{min}} \leq L \), the loop beginning at line 9 runs for at most L^Q iterations. (This bound can be improved to n^{O(Q)}; see the remark below. However, we use this loose bound as this keeps the argument simpler, without affecting the qualitative nature of our result.) Finally, by Fact 2.2, there are at most n^{2/δ} possibilities for each δ-restricted profile \( \pi_q \). Thus, the loop beginning at line 11 runs for at most n^{Q^2/δ} iterations.

Putting it all together, the algorithm makes at most 2n^{Q/δ^2} + 2Q/δ^2 L^Q recursive calls to itself, with subproblems of size at most n/2 each and uses an additional poly(n) processing time. Recalling that Q = 1 + \lfloor \log \max_i \{w_i/\rho_i\} \rfloor ≤ O(\log(Ln/δ)) and using L ≤ 2^{polylog(n)}, we see that the overall running time is bounded by 2^{polylog(n)}, as claimed. □

**Remark.** The number of steps at line 9 can be improved to \( n^{O(Q)} \) from \( L^Q \). Consider an arbitrary class q. By definition, \( \rho_i \in [w_i/2^q, w_i/2^{q−1}] \) for demand i in class q. Thus, it suffices to consider only those \( h_q^* \) in line 9 that are an integer multiple of 1/2^q instead of \( \rho_{\text{min}} \). Moreover, since \( w_i \in [1, n/\epsilon^2] \) and there are most n demands in the instance, it follows that the cumulative
Thus, we have \( e \) choices to consider for each \( q \) and hence at most \( n^{O(\Delta)} \) steps at line 9.

**Theorem 4.2.** Let \( O \) be an optimal solution to the given UFP instance and \( \delta \) be a small enough positive real such that \( 1/\delta \) is an integer. Algorithm LINE-UFP-RECURSIVE returns a feasible solution with profit at least \( (1 - 13\delta)w(O) \).

**Proof.** We proceed by induction on \( |S| \). The algorithm clearly returns an optimal solution if \( |S| \leq 1 \), so we focus on the case \( |S| > 1 \).

Consider the edge \( e \) identified by the algorithm in line 3 and the sets \( S_0 \) of class-\( q \) demands that span \( e \). Let \( A_q := S_0 \cap O \) be the subset of \( S_0 \) routed by \( O \) and let \( h_q := \text{load}(A_q, e^\ast) \). We assume, without loss of generality, that \( h_q > 0 \), i.e., that \( A_q \neq \emptyset \). Let us define the sets

\[
A_q,\text{large} := \{ i \in A_q : \rho_i > \delta^2 h_q \},
\]

\[
A_q,\text{small} := \{ i \in A_q : \rho_i \leq \delta^2 h_q \}.
\]

Since the demands in \( A_q,\text{large} \) all span \( e^\ast \) and have total load at most \( h_q \) on \( e^\ast \), there can be at most \( 1/\delta^2 \) of them; so the algorithm will eventually set \( T_q = A_q,\text{large} \), for each \( q \). From now on, we focus on only this iteration of the loop beginning at line 5.

Consider any arbitrary \( q \) and apply Lemma 3.1 to the set \( A_q,\text{small} \), noting that \( B \geq \delta^2 h_q \) is a bound on the \( \rho_i \)'s. The algorithm will eventually pick \( h_q \) to be the largest integer multiple of \( \rho_{\text{min}} \) not exceeding \( \text{load}(A_q, e^\ast) \) and \( \pi_q \) to be the \( \delta \)-restricted profile whose existence is guaranteed by Lemma 3.1. Let us concentrate on these choices of \( h_q \) and \( \pi_q \). We have

\[
h_q + \text{load}(T_q, e^\ast) \leq h_q < h_q + \rho_{\text{min}} + \text{load}(T_q, e^\ast).
\]

Let \( W \) be the maximum possible profit of a subset of \( A_q,\text{small} \) that fits profile \( \pi_q \). Then

\[
W \geq w(A_q,\text{small}) - 2^{q+1}(\delta h_q + B) \\
\geq w(A_q,\text{small}) - 2^{q+1}(\delta + \delta^2)h_q,
\]

where we used the left inequality in (4.4). Now consider the construction of the set \( S_q,\text{small} \) in line 10; the right inequality in (4.4) ensures that it is a superset of \( A_q,\text{small} \). Moreover, for any \( i \in S_q,\text{small} \), we have \( \rho_i \leq \delta^2 h_q + \rho_{\text{min}} \leq 2\delta^2 h_q \), using the left inequality in (4.4). Lemma 3.2, applied with the bound \( B = 2\delta^2 h_q \), says that when the algorithm callsPILE-PACK with this profile \( \pi_q \) and the set of demands in \( S_q,\text{small} \), it will obtain a set \( U_q \) with

\[
w(U_q) \geq W - 2^{q+1}(2\delta^2 h_q)/\delta \\
\geq w(A_q,\text{small}) - 2^{q+1}(3\delta + \delta^2)h_q \\
\geq w(A_q,\text{small}) - 13\delta \cdot 2^{q-1}h_q,
\]

where the final inequality uses the assumption that \( \delta \) is small enough.

Thus, we have \( \text{load}(U_q) + \text{load}(U_q) \geq w(A_q) - 13\delta \cdot 2^{q-1}h_q \geq (1 - 13\delta)w(A_q) \), since each demand in \( S_q \) has profit density at least \( 2^{q-1} \). Since \( q \) was arbitrary, the algorithm will eventually satisfy these conditions for all \( q \).

Lemmas 3.1 and 3.2 also guarantee that for each \( q \), \( \text{profit}(U_q) \leq \pi_q \leq \text{profit}(A_q,\text{small}) \). Therefore, routing \( T \cup U \) leaves at least as much residual capacity on each edge as doing routing \( U \). It follows that the two recursive calls work on graphs with “sufficient” residual capacity. Let \( O_L \) and \( O_R \) denote the subsets of \( O \) consisting of demands that lie to the left of \( e \) and to the right of \( e \), respectively. By induction hypothesis, the recursive calls return sets \( L \) and \( R \) with \( |L| \geq 1 - 13\delta|O_L| \) and \( |R| \geq 1 - 13\delta|O_R| \). Therefore, Algorithm LINE-UFP-RECURSIVE will eventually record a solution with profit at least \( (1 - 13\delta)w(O) \).

**Theorem 4.3.** *(First Part of the Main Theorem).* There is a quasi-polynomial time approximation scheme for UFP on line graphs, provided all capacities and resource requirements are integers bounded by \( 2^{O \log(n)} \).

**Proof.** This follows immediately from Theorems 4.1 and 4.2, together with our observation at the beginning of Section 2 that an upper bound on the profits is unnecessary for obtaining a \((1 + \epsilon)\)-approximation.

5. Extension to Cycles

We show that the techniques in the previous section can be extended to the case of cycles. We first consider the simple case of a directed cycle in which all the edges are directed in the same direction. In this case, the result follows directly from the results for a line. Consider an edge \( e \) and partition the set of demands \( D \) into two types: those that load the edge \( e \) (call these \( D_1 \)) and those that do not. Let \( O \) denote some optimal solution. The demands in \( D_1 \cap O \) form a pile at \( e \). For each profit density class, we guess the \( \mathcal{O}(1) \) large demands and the “right” \( \rho \)-restricted profile and pack this profile almost optimally. This gives a packing of a subset \( S \subseteq D_1 \) with total profit almost equal to that of \( D \setminus O \). Moreover, this packing does not interfere with \( D \setminus D_1 \). We then move the edge \( e \), adjust the edge capacities to account for demands in \( S \) and restrict our set of demands to \( D \setminus D_1 \). This yields an instance of UFP on a line which can now be solved using our earlier techniques.

We now consider the harder case of an undirected cycle. The crucial difference in this case is that every demand has two choices of paths along which it can be routed. Thus, in addition to deciding which demands to select, the algorithm also needs to specify the route for each selected demand. As in the case of a line, we will give an algorithm that has \( O(\log n) \) levels of recursion and has a quasi-polynomially large branching factor at each level. In particular, given a cycle of length \( m \) and a set of demands \( D \), the algorithm first partitions the demands into three sets \( D_0, D_1 \) and \( D_2 \). It packs an almost optimal subset of \( D_0 \) in a profile chosen out of quasi-polynomially many candidate profiles, and then recurses on two cycles \( C_1 \) and \( C_2 \) each of length \( m/2 \) with demand sets \( D_1 \) and \( D_2 \) respectively. We now describe the details.

Let \( C \) be a cycle on vertices \( 0, \ldots, m - 1 \) and let \( e_i \) denote the edge \( \{i, i+1\} \), where the indices are considered modulo \( m \). For convenience we will assume that \( m \) is even, and will use 0 and \( m \) interchangeably. For \( i = 1, \ldots, n \), the \( i \)-th demand is specified by \( (s_i, t_i, \rho_i, w_i) \). As before, we can assume that \( m \leq 2n \).

We will also assume that the edge capacities and demands are integers bounded by \( L \), which is quasi-polynomial in \( n \). Let \( Q = O(\log(n)) \) denote the number of profit density classes.

Consider the two antipodal edges \( e_0 \) and \( e_{m/2} \). We partition the vertices into two sets \( V_1 = \{1, \ldots, m/2\} \) and \( V_2 = \{m/2 + 1, \ldots, m\} \). We partition the set of demands \( D \) into three sets based on the locations of their end-points. Let \( D_1 = \{i : s_i \in V_1, t_i \in V_2\} \) be the set of demands that have both end-points in \( V_1 \). Similarly, let \( D_2 \) be the set of demands with both end-points in \( V_2 \). Let \( D_0 \) be the rest of the demands. The demands in \( D_0 \) have exactly one end-point in \( V_1 \) and one in \( V_2 \). We refer the reader to Figure 2 for greater clarity.

Let \( O \) be some fixed optimal solution. Note that in addition to specifying a subset of demands, \( O \) also specifies a routing for each demand in the subset. We will abuse notation somewhat and also use \( O \) to refer to the subset of demands in the solution \( O \). Consider an arbitrary solution \( A \) and let \( S \subseteq A \) be an arbitrary subset of demands in \( A \). We use \( \text{load}(S, A, e) \) to denote the load on edge \( e \).
due to the demands in \( S \) when they are routed as in \( A \). Define the \( m \)-dimensional vector

\[
\text{prof}(S, A) := (\text{load}(S, A, e_0), \text{load}(S, A, e_1), \ldots, \text{load}(S, A, e_{m-1}))
\]

We will be interested in the profile \( \text{prof}(O \cap D_0, O) \). The following lemma shows that any profile due to a subset of demands in \( D_0 \) can be approximated.

**Lemma 5.1.** Let \( S \subseteq D_0 \) be a subset of demands and \( T \) be some solution that defines a feasible routing of \( S \). Then, for any sufficiently small parameter \( \delta > 0 \), there exists another simpler profile \( \pi \) such that

1. \( \pi \leq \text{prof}(S, T) \).
2. There are at most \( O(Q/\delta^3) \) distinct non-zero entries in \( \pi \), where \( Q \) is the number of distinct profit density classes.
3. There exists a collection of demands \( S' \subseteq D_0 \) with a suitable routing \( T' \) such that \( \text{prof}(S', T') \leq \pi \) and \( w(S') \geq (1 - c\delta)w(S) \), where \( c \) is some fixed constant independent of \( \delta \).

Moreover, the set \( S' \) can be found in time quasi-polynomial in \( n \).

We now consider the demands in \( D_1 \) and \( D_2 \). Let \( q_1 \) denote the total load due to demands in \( D_1 \cap O \) that are routed along the edges \( e_{m/2}, e_{m/2+1}, \ldots, e_m = e_0 \) (i.e., these are the demands that are not routed along their shortest paths). Note that these demands load each edge \( e_{m/2}, \ldots, e_0 \) by exactly \( q_1 \). Similarly, let \( q_2 \) denote the total load due to demands in \( D_2 \cap O \) that are routed along \( e_0, e_1, \ldots, e_{m/2} \).

Our algorithm is the following: We guess the “right” approximate profile \( \pi \) and also guess the quantities \( q_1 \) and \( q_2 \). By Lemma 5.1, we find a high profit subset of \( D_0 \) and pack these demands in \( \pi \). We reduce the capacities on edges \( e_0, \ldots, e_{m/2} \) by \( q_2 \) and on the edges \( e_{m/2}, e_{m/2+1}, \ldots, e_0 \) by \( q_1 \).

We now remove the edges \( e_0 \) and \( e_{m/2} \) from the cycle \( C \), and introduce two new edges \( e' = \{m/2, 1\} \) and \( e'' = \{0, m/2 + 1\} \). We assign \( e' \) a capacity of \( q_1 \) and \( e'' \) a capacity of \( q_2 \). This yields two vertex-disjoint cycles, \( C_1 = (e', e_1, \ldots, e_{m/2-1}) \) and \( C_2 = (e'', e_{m/2+1}, \ldots, e_{m-1}) \), each of length \( m/2 \). We recurse on \( C_1 \) with the instance \( D_1 \) and on \( C_2 \) with the instance \( D_2 \). Figure 2 illustrates this step.

We now show that this algorithm has the desired properties.

**Theorem 5.2.** (Second Part of the Main Theorem). The algorithm above gives a quasi-polynomial time approximation scheme for UFP on a cycle, provided all capacities and resource requirements are integers bounded by \( 2^{\text{polylog}(n)} \).

**Proof.** There are \( \log m = O(\log n) \) levels of recursion and only quasi-polynomially many choices for the profile \( \pi \) and the quantities \( q_1 \) and \( q_2 \) at each level. It follows that running time of the algorithm is quasi-polynomial in \( n \).

Let \( O \) be some fixed optimal solution and \( \varepsilon > 0 \) be an arbitrarily small parameter. By Lemma 5.1, the algorithm finds a subset \( S \subseteq D_0 \) with profit at least \((1 - \varepsilon)w(D_0 \cap O) \). Moreover, by construction of \( C_1 \) and \( C_2 \), \( D_1 \cap O \) is a feasible solution for \( C_1 \), \( D_2 \cap O \) is a feasible solution for \( C_2 \), and both these sets can be packed together feasibly with \( S \). Since \( D_0, D_1 \), and \( D_2 \) are mutually disjoint, a simple inductive proof shows that the total profit obtained by the algorithm is at least \((1 - \varepsilon)w(O)\). \( \square \)

It remains to prove Lemma 5.1. The proof is similar in spirit to that for the case of a line, but more involved. In fact, we will use the results of Schrijver, Seymour and Winkler [18] to simplify our arguments. The following result is implicit in their work (we give a proof sketch since this result is not explicitly stated).

**Theorem 5.3.** Consider UFP on a cycle with \( m \) edges and let \( O \) be an optimal solution. Then there is a polynomial time algorithm that finds a solution \( O' \) such that \( w(O') \geq w(O) - (m^2/2)w_{\max} \), where \( w_{\max} \) is the largest profit of any demand.
PROOF SKETCH. Consider a linear programming formulation with two variables $x_{d,1}$ and $x_{d,2}$ per demand, where $x_{d,1}$ indicates the fraction of demand $d$ that is routed in the clockwise direction and $x_{d,2}$ is the fraction that is routed in the counter-clockwise direction. There are natural sets of constraints to enforce that no capacity is violated and another set of constraints of the type $x_{d,1} + x_{d,2} \leq 1$, to enforce that no demand is used more than once.

Proposition 4.2 in [18] implies that in any vertex solution to the linear program above, there at most $m(m-1)/2$ demands such that $0 < x_{d,1} + x_{d,2} < 1$. Moreover, as shown in the argument after the proof of Proposition 4.2, any LP solution can be transformed without any loss of profit into another one where at most $m/2$ demands are “split” along two routes.

By throwing away the demands that are either routed fractionally or split, the result follows trivially. \qed

We now state a useful property of $\delta$-restricted profiles. Let $\pi$ be a $\delta$-restricted profile on a line with peak at $e$, and let $D$ be a collection of class $q$ demands such that each of them spans $e$. Define $O(\pi, D)$ to be the maximum profit of over all subsets $T \subseteq D$ such that $\text{prof}(T) \leq \pi$. Let $D^-(\pi)$ denote the instance obtained from $D$ by extending each demand such that its end-points align with the steps of the profile $\pi$. Formally, let

$$\pi = RP_S(c; h; x_1, \ldots, x_{1/\delta} + y_1, \ldots, y_{1/\delta})$$

(as in Definition 2.1) and let $s_i, t_i$ be the end-points of $i \in D$ and suppose $s_i < t_i$. Let $\alpha$ be the index such that $x_{\alpha} \leq s_i < x_{\alpha+1}$ and $\beta$ be such that $y_{\beta+1} \leq t_i \leq y_{\beta}$; we define the new end-points $s^+_{i} = x_{\alpha}$ and $t^+_{i} = y_{\beta}$. We then define the new instance $D^+(\pi)$ to consist of $\{(s^+_{i}, t^+_{i}, \rho_i, w_i)\} \in D$. Similarly, let $D^-(\pi)$ denote the instance obtained from $D$ by shrinking each demand to align with the steps of the $\pi$. Thus, if $x_{\alpha-1} < s_i < x_{\alpha}$ and $y_{\beta} \leq t_i < y_{\beta+1}$, we set $s^+_{i} = x_{\alpha}$ and $t^+_{i} = y_{\beta}$. We claim that the optimal solutions for the instances $D^+(\pi)$ and $D^-(\pi)$ in profile $\pi$ are not too far from $D$.

**Lemma 5.4.** For $D^+(\pi)$ and $D^-(\pi)$ as defined above, we have

1. $O(\pi, D) - 2^{q+1} (\delta h + B) \leq O(\pi, D^+(\pi)) \leq O(\pi, D)$.
2. $O(\pi, D) \leq O(\pi, D^-(\pi)) \leq O(\pi, D) + 2^{q+1} (\delta h + B)$.

where $B$ is the maximum profit of any demand.

**Proof.** The result follows directly using the argument in the proof of Lemma 3.1. \qed

We are now ready to prove Lemma 5.1.

**Proof of Lemma 5.1.** Any demand in $S$ spans exactly one of $e_0$ and $e_{m/2}$. Let $A_q \subseteq S$ be the set of class $q$ demands that span $e_0$. Clearly, $A_q$ forms a pile at $e_0$. Let $h_q$ denote load$(A_q, e_0)$. We define the sets,

$$A_{q, \text{large}} := \{ i \in A_q : \rho_i > \delta^3 h_q \},$$
$$A_{q, \text{small}} := \{ i \in A_q : \rho_i \leq \delta^3 h_q \}.$$

For each class $q$, we use Lemma 3.1 to approximate the profile prof$(A_{q, \text{small}})$ by a $\delta$-restricted profile $\pi_{q, \text{small}}$. By Lemma 3.1 there is a set of demands $S_{q, \text{small}} \subseteq A_{q, \text{small}}$ such that prof$(S_{q, \text{small}}) \leq \pi_{q, \text{small}}$ and $w(A_{q, \text{small}} \setminus S_{q, \text{small}}) \leq 2^{q+1} (\delta h_q + \delta^3 h_q) \leq 8 \delta (2^{q+1} h_q) \leq 8 \delta w(A_q)$.

Similarly, we define $\tilde{A}_q$ to be the set of class $q$ demands that span the edge $e_{m/2}$, and apply an analogous procedure to obtain the profiles $\tilde{\pi}_{q, \text{small}}$ and the sets $\tilde{S}_q$. Let

$$\pi := \sum_{q=1}^{\Omega(\text{prof}(A_{q, \text{large}}) + \text{prof}(\tilde{A}_{q, \text{large}}) + \pi_{q, \text{small}} + \tilde{\pi}_{q, \text{small}})}.$$

It follows that $\pi$ has at most $O(\delta^3)$ distinct entries and at least $(1 - \delta\delta)$ fraction of the profit of $S$ can be placed feasibly in $\pi$.

We now show how to efficiently find a high profit subset of demands that can be placed feasibly in $\pi$. Note that to obtain $\pi$, the algorithm needs to guess the right $A_{q, \text{large}}, \tilde{A}_{q, \text{large}}, \pi_{q, \text{small}}$ and $\tilde{\pi}_{q, \text{small}}$ for each $q = 1, \ldots, Q$. It remains to show that we find a sufficiently profitable set of small demands.

Consider a profit density class $q$. Let $h_{q, \text{small}}$ and $\tilde{h}_{q, \text{small}}$ denote the heights of $\pi_{q, \text{small}}$ and $\tilde{\pi}_{q, \text{small}}$ respectively. Let $\pi_q = \pi_{q, \text{small}} + \tilde{\pi}_{q, \text{small}}$. Let $D_{0, q, \text{small}}$ denote the set of class $q$ demands in $D_0$ that have resource requirement at most $\delta^3 \max\{h_q, \tilde{h}_q\}$. Consider the UFP instance where the demand set is $D_{0, q, \text{small}}$ and the edge capacity vector is equal to $\pi_q$. Clearly, $\tilde{S}_{q, \text{small}} \cup \tilde{S}_q$ is a feasible solution for this instance. Let $w^*$ denote the profit of this solution. Consider the instance $D_{0, q, \text{small}}$ obtained from $D_{0, q, \text{small}}$ by moving the end-points of the demands to align with closest “step” of the profile $\pi_q$. By Lemma 5.4 it follows that there is feasible solution to this modified instance with value $w'$ where

$$w' \geq w^* - 2^{q+1} (\delta h_{q, \text{small}} + \delta \tilde{h}_{q, \text{small}} + 2 \delta^3 \max\{h_q, \tilde{h}_q\}) \geq w^* - 2^{q+1} (45 \max\{h_q, \tilde{h}_q\}) \geq w^* - 16 \delta (w(A_q) + w(\tilde{A}_q)) \quad (5.5)$$

Consider the UFP instance with demands $D_{q, \text{small}}$ and capacities $\pi_q$. All the demands in this instance aligned with the steps of $\pi_q$. As $\pi_q$ has at most $4/\delta + 2$ steps, this problem is combinatorially equivalent to that on a cycle with $m = 4/\delta + 2 \leq 6/\delta$ edges. By Theorem 5.3, there is a polynomial time procedure to find a solution with profit at least $w'' = w^* - (18/\delta^2) 2^{q+1} \delta^3 \max\{h_q, \tilde{h}_q\} \geq w^* - 72 \delta (w(A_q) + w(\tilde{A}_q))$. Applying Lemma 5.4 again and by an analysis identical to the one in inequality (5.5) above, we obtain a solution with profit at least $w''' = w^* - 16 \delta (w(A_q) + w(\tilde{A}_q))$.

Hence, overall, the algorithm finds a packing of class $q$ demands that completely lies inside the profile $\pi_q + \text{prof}(A_{q, \text{large}}) + \text{prof}(\tilde{A}_{q, \text{large}})$ and has profit at least $(1 - O(\delta))(w(A_q) + w(\tilde{A}_q))$, which implies the desired result. \qed

6. CONCLUDING REMARKS

We have provided a quasi-PTAS for UFP on line and cycle graphs, thereby virtually ruling out an APX-hardness result for the problem. Unlike most earlier work, we do not require a no-bottleneck assumption.

An immediate open question is whether there is in fact a PTAS for the same problem. At first glance it may appear that the reason why our algorithm has only a quasi-polynomial worst case running time is that it has to try out a large number of ways of allocating resources amongst the different profit classes. However, the reason seems to be more fundamental and inherent in our approach: the algorithm involves a $\Theta(\log n)$-depth recursion with a super-constant branching factor. In particular, we do not know how to obtain a PTAS even in the case when there is a single profit class. Moreover, in the special case where the capacities and demands are bounded by a constant (i.e. $L = O(1)$), our algorithm still takes quasi-polynomial time, whereas a simple dynamic program can be used to find an exact solution in polynomial time. It appears that
some further insight is needed to obtain a PTAS.
Another obvious open question is whether our result extends to more classes of graphs, such as stars or trees of depth 2. Determining the simplest class of graphs for which UFP is APX-hard remains an interesting open problem.

7. REFERENCES