Masters Dissertation

# Approximation Algorithms for Stochastic Unsplittable Flow Problems

Archit Karandikar

CMU-CS-15-142
December 2015

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Submitted in partial fulfillment of the requirements
for the Degree of Master of Science

Thesis Committee:
Anupam Gupta (Chair)
R Ravi
Danny Sleator

**Keywords:** Approximation Algorithms, Stochastic Unsplittable Flow Problem, Linear Programming Relaxations, Randomized Rounding, Adaptivity Gap, Totally Unimodular Matrices

# Acknowledgements

I am grateful to all the people who helped me complete my masters dissertation.

Thanks to my advisor, Anupam Gupta, for all the priceless guidance, advice, encouragement and support. I am fortunate to have such a capable and helpful advisor.

Thanks to R Ravi for analyzing and providing insights into the problems addressed by this dissertation.

Thanks to Danny Sleator for helping with my thesis and for providing me a strong foundation in Theory.

Thanks to my parents, Vikrant Karandikar and Ashlesha Karandikar, for providing unconditional love, encouragement and support in all my efforts, including my research. Thanks to my grandparents, late Dr. Indumati Karandikar, late Dr. Laxman Karandikar, Rohini Menavlikar and late Madhav Menavlikar for your love and for being a source of inspiration an strength.

Thanks to my family members and my teachers, especially Prof. M. Prakash, Prateek Karandikar and Rajeeva Karandikar for providing me with the mathematical foundation central to all of my academic endeavours.

I am greatly indebted to so many friends, too numerous to name here, for being a ceaseless source of strength and happiness throughout my masters program.

*To Dr. Indumati Karandikar,*

# Abstract

In this thesis, we present approximation algorithms for variants and special cases of the Stochastic Unsplittable Flow Problem (hereafter, sUfp). The objective of the sUfp is to optimally schedule tasks from a given set of tasks on an underlying graph, each of which is specified by a stochastic demand, a payoff and a pair of vertices between which it must be scheduled. Even very special cases of the sUfp, such as the sUfp on a single-edge graph with deterministic demands, are known to be NP-hard.

We present new polytime constant-factor approximation algorithms for the sUfp on Paths and Trees and for extensions where each task may correspond to more than two vertices. We consider two settings - a special-case in which all the capacities are equal and the general-case in which the capacities are arbitrary. In dealing with arbitrary capacities, we assume that the distribution underlying the demand for each task has maximum attainable value less than or equal to the least capacity among the edges in the graph, operating under what is known as the No Bottleneck Assumption.

Our approximation algorithms are obtained by combining approximations for instances in which all tasks are small and for those in which all tasks are large. They provide an approximation to a linear programming relaxation and hence may perform significantly better in practice than the guarantees we establish. These algorithms do not make decisions based on results of previously instantiated tasks, and are classified as non-adaptive algorithms, which leads to the result that the adaptivity gaps for these problems are bounded by a constant.

# Contents

# Chapter 1

# An Introduction to the Stochastic Unsplittable Flow Problem

## 1.1 Overview

The Unsplittable Flow Problem (hereafter, the Ufp) deals with the optimal allocation of a resource (such as bandwidth) at each communication link of a network to a set of tasks each having a resource demand between pairs of nodes on that network and a payoff. The Ufp has been a subject of a lot of research interest in recent times, since it arises naturally in applications such as bandwidth allocation, resource-constrained scheduling and packing.

Motivated by the fact that in several such applications, the amount of resource required by an entity is not known deterministically beforehand, we consider the Stochastic Ufp (hereafter, the sUfp). In the sUfp, the actual resource demand of each task itself is unknown in advance, but we assume that the demand is drawn from a probability distribution. We will address instances of the sUfp in which the network is a tree. We also consider extensions of the sUfp on trees where each task must be scheduled between multiple nodes. As is standard in addressing the Ufp we will assume that the maximum attainable resource demand is at most the minimum link capacity. This is called the *No Bottleneck Assumption* (hereafter, Nba).

Even very special cases of Ufp, which is itself a special case of the sUfp, are NP-hard. When the graph representing the network is a single edge, the Ufp is equivalent to the Knapsack problem. When all demands, payoffs and all capacities are 1, the Ufp is equivalent to the Maximum Edge-Disjoint Paths problem. When all demands are 1, the Ufp is equivalent to the Integer Multicommodity Flow problem.

## 1.2 Problem Specifications

In the sUfp, we are given a graph $G = (V, E)$ and a set of tasks $T$. Let the number of tasks be denoted by $n$ and let $G$ have $p$ vertices and $m$ edges. Each edge $e \in E$ corresponds to a resource of which $c_e$ units are available. We call $c_e$ the capacity of edge $e$. Each task $t \in T$ is specified by a pair of distinct vertices $\{a_t, b_t\} \in V$ and can be scheduled along any path between $a_t$ and $b_t$. The size of the task $t$ is given by a positive random variable $S_t$, whose distribution is given to the algorithm as input. [1] If task $t$ is *scheduled*, we specify a path $P_t$ for it. [2] Task $t$ then consumes $S_t$ amount of resource on each edge along path $P_t$. As a result of scheduling task $t$, the residual capacity of edge $e$, changes as: $\widehat{c}_e \leftarrow \max(0, \widehat{c}_e - S_t)$. A task is *successfully scheduled* if $S_t \leq \min_{e \in P_t} \widehat{c}_e$ and we get a payoff of $v_t$, else it is *unsuccessfully scheduled* and we get no payoff. Observe that even if the job is unsuccessful, the capacity along every edge in $P_t$ is consumed to extent $S_t$ or to complete consumption, whichever occurs first. The objective is to find a scheduling strategy to maximize the expected payoff.

[1] As will become apparent, we need to know only some statistics of this distribution for our algorithms.

[2] In this work, we will deal only with the sUfp on Trees and its extensions. For our purposes the path $P_t$ for every task $t$ is uniquely defined. For the extensions we consider, $P_t$ may be a subtree instead of a path.

Note that each task can be scheduled at most once. The random variable $S_t$ is instantiated only when task $t$ is scheduled and all scheduling decisions are irrevocable. Also note that since the sizes $S_t$ of all tasks $t$ are positive random variables, once an edge has zero residual capacity, it cannot be a part of any successfully scheduled task.

We consider some special cases of the sUfp based on restricting the graph:

- When the graph $G$ is a tree, the problem is called the *Stochastic Unsplittable Flow Problem on a Tree*, or the sUfpTree. In this case, we imagine the graph $G$ as being rooted at some node $r \in V$. The unique $a_t$-$b_t$ path is now denoted by $P_t$. The *depth of node* $v$ is the number of edges on the unique $v$-$r$ path. The *depth of a path* is the depth of the least common ancestor (hereafter, *lca*) of the endpoints of the path. Equivalently, it is the depth of the least-depth node on the path. The *depth of a task* is the depth of the path corresponding to it. Assume that the tasks are numbered from 1 to $n$ in non-decreasing order of depth, that for all tasks $t$, the terminals $a_t$ and $b_t$ are leaves in $G$, and that the root has a single child. All these assumptions are without loss of generality.

  If in an instance of the sUfpTree, all edge capacities are equal, we get the *Stochastic Resource Allocation Problem on a Tree* (hereafter, the sRapTree). We assume by scaling in dealing with the sRapTree that all capacities are 1.

- When the graph G is a path, the problem is called the *Stochastic Unsplittable Flow Problem on a Path*, or the sUfpPath. The vertices are numbered from 1 to $m+1$ along the path from left to right, and edges of the path from 1 to $m$. As in the sUfpTree, the unique $a_t$-$b_t$ path $P_t$ consists of edges in the set $[a_t, b_t)$. We assume that the tasks are sorted by their left endpoints, i.e., $a_1 \leq \ldots \leq a_n$. Note that this is same as sorting the tasks in non-decreasing order of depth if we consider vertex 1 to be the root. Unlike the sUfpTree, we do not assume in the sRapTree that $a_t$ and $b_t$ are leaves in $G$.

  If in an instance of the sUfpPath, all edge capacities are equal, we get the *Stochastic Resource Allocation Problem on a Path* (hereafter, the sRapPath). We assume by scaling for the sRapPath, as we did for the sRapTree that all capacities are 1.

  Note that the sUfpPath and the sRapPath are special cases of the sUfpTree and the sRapTree respectively.

We can also generalize the idea of scheduling pairs $\{a_t, b_t\}$ to scheduling sets of terminals. Specifically, when addressing the sUfpTree and the sRapTree, we generalize our results as follows:

- A tree $T'$ rooted at $r'$ is called a *$k$-spider* if $T'$ has at most $k$ leaves, and the least common ancestor of any two nodes $u, v$ in it is either the root $r'$, or one of $\{u, v\}$. Equivalently, a *$k$-spider* is a set of at most $k$ disjoint "downward" paths emanating from the root $r'$. Given a tree $T$, call a set $D_t \subseteq V$ a *$k$-spider set* if the subtree induced by these vertices is a *$k$-spider* set. Note that paths are *2-spiders*. (An alternative definition is as follows: given $D_t$, let $D'_t$ be a minimal subset which indices the same subtree as $D_t$. Then $D_t$ is a *$k$-spider* if $\mid D'_t \mid \leq k$ and $D'_t$ has the Helly-type condition that the *lca* of any pair of its elements is the same as the *lca* of the entire set.)

  The generalization of the sUfpTree where each task $t$ corresponds to a *$k$-spider* set $D_t$, is called the *Stochastic Unsplittable Flow Problem on a Tree with $k$-spiders* (hereafter, the sUfpTree-$k$Spider). [3]

  Along similar lines, the generalization of the sRapTree where each task $t$ corresponds to a *$k$-spider* set $D_t$, is called the *Stochastic Resource Allocation Problem on a Tree with $k$-spiders* (hereafter, the sRapTree-$k$Spider). [3] Again, we assume by scaling for the sRapTree-$k$Spider that all capacities are 1.

  For the sUfpTree-$k$Spider and the sRapTree-$k$Spider, let the *$k$-spider* induced by task $t$ be denoted by $P_t$, and the tasks be numbered from $1, \ldots, n$ in non-decreasing order of the depth of the root of their corresponding *$k$-spiders*. As in the sUfpTree, we assume that the terminals in set $D_t$ for each task $t$ are leaves and that the root has a single child.

  In the process of obtaining an approximation algorithm for the sUfpTree-$k$Spider, we will come up with an approximation algorithm under the Nba for the version of this problem where all demands are deterministic. We will denote this problem by the UfpTree-$k$Spider.

[3] We include $k$ in the problem name and abbreviation because the approximation factor we obtain depends on $k$.

Note that the sUfpTree and the sRapTree are equivalent to the sUfpTree-2Spider and the sRapTree-2Spider respectively. Also note that the sUfpPath and the sRapPath are special cases of the sUfpTree-1Spider and the sRapTree-1Spider respectively.

- Finally, we also consider the generalization where $G$ is a rooted tree and each task $t$ corresponds to a subtree $P_t$ of $G$. Our results will depend on the upper bound on the number of children of each node in $G$. We consider the case where all edge capacities are equal, and as before, scaled to 1. We refer to this problem as the *sRap on $k$-ary trees with subtrees* or the sRap$k$Tree-Subtree. As in the sUfpTree, we assume that the tasks are numbered from $1, \ldots, n$ in non-decreasing order of the depth of the root of their corresponding subtrees.

  One of our intermediate results is for the extension of the sRapTree where each task corresponds to a $k$-ary subtree rooted at its vertex of least depth in $G$. We will refer to this problem as the sRapTree-$k$Subtree.

Note that all problems mentioned here are special cases of the *Stochastic Unsplittable Flow Problem on Trees with Subtrees*, denoted hereafter by the sUfpTree-Subtree, which is the generalization of the sUfpTree where each task corresponds to a subtree.

### 1.2.1 The No-Bottleneck Assumption (Nba)

Improved results for unsplittable-flow problems have often been obtained under the Nba. In the stochastic context, the Nba says that for each task $t \in T$, the domain of the size of random variable $S_t$ is $(0, c_{\min}]$ where $c_{\min} = \min_{e \in E} c_e$. By scaling we assume that $c_{\min} = 1$, and hence the r.v.s $S_t \in (0, 1]$. We operate under the Nba for the sUfpPath and the sUfpTree-$k$Spider.

### 1.2.2 Problem Hierarchy

We have defined a lot of different problems in §1.2. In this section we provide figures to depict the setting for each problem and a schematic for the hierarchy of problems to help the reader get acquainted with the notation we introduced.

The following figure depicts the setting for each of the special cases of the sUfp relevant to our work. Specifically, it shows the nature of the graph and the nature of the subgraph that each task corresponds to. Each of the four subfigures corresponds to a few special cases and depicts the graph $G$ and the subgraph $P_t$ corresponding to a single task $t$ for the relevant setting. The edges in $P_t$ are shown in red and those not in $P_t$ are shown in black. For the two figures on the right, the depiction is valid for $k \geq 3$.

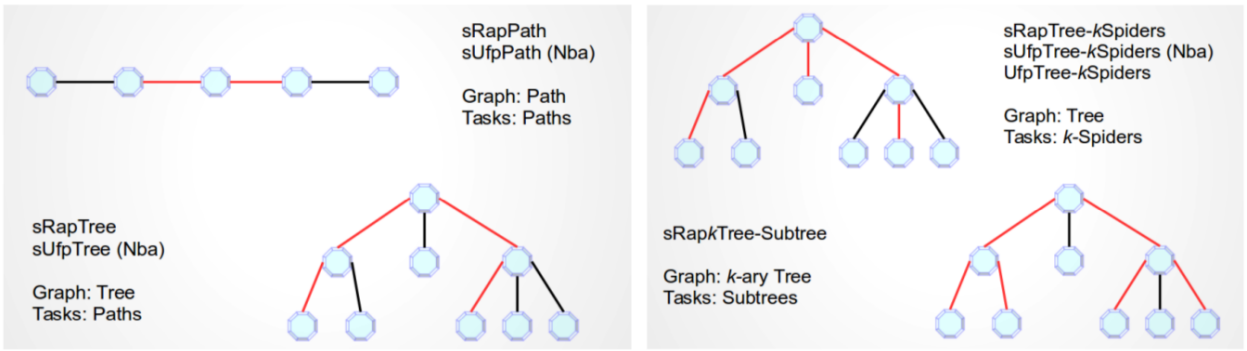


Figure 1.1: Relevant Special Cases of the sUfp

The problem hierarchy is represented below with a schematic which shows *generalization* →*special case* relations between the problems. Relations implied by transitivity are not shown.

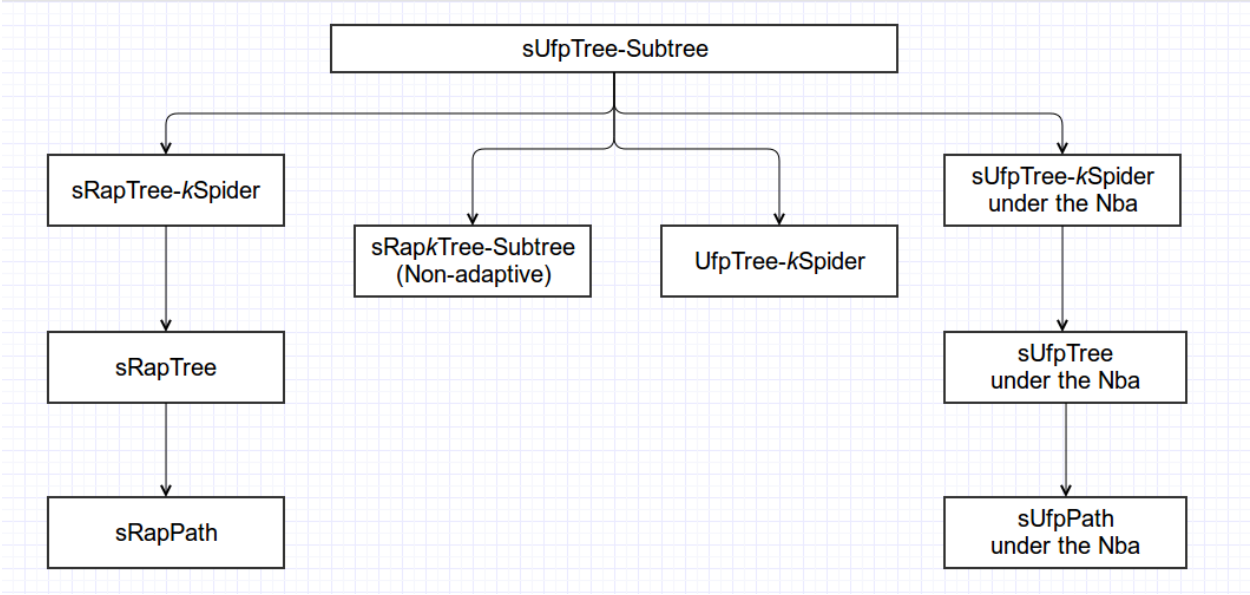


Figure 1.2: Problem Hierarchy

We obtain approximation results for each of the problems represented in Figure 1.2 besides the sUfpTree-Subtree. For the sRap$k$Tree-Subtree, we only obtain an approximation to non-adaptive algorithms (defined in §1.6) and this is shown in the figure. The only relevant problem not shown in this figure is the sRapTree-$k$Subtree because we only show an intermediate result and do not obtain an approximation for this problem. The sRapTree-$k$Subtree problem is a generalization of all the sRap variants (every problem in the left half of Figure 1.2) and the result we show for it will be used by all of these variants.

## 1.3 Relationship to Previous Work

**Unsplittable Flow Problems.** As stated above, there is much research related to the Ufp. Here we just state the approximation results to Ufp that are directly relevant to our work. The Ufp on general graphs generalizes the Maximum Edge-Disjoint Path problem. This problem is $\Omega(\sqrt{|E|})$-hard for directed graphs [GKR+03], and a matching $O(\sqrt{|E|})$-approximation algorithm was given by Kleinberg [Kle98]. Other results for this problem are given by Srinivasan [Sri97], Kolliopoulos and Stein [KS04], Kolman and Scheideler [KS06], Chekuri et al. [CKS06], and others. The hardness of the Maximum Edge-Disjoint Path for the undirected case is weaker, and closing the gap between the upper and lower bounds in this case remains an interested direction for research.

Given the hardness of Ufp for general graphs, there is much interest in Ufp on special classes of graphs. The Ufp on Trees in APX-hard. Calinescu, Chakrabarti, Karloff and Rabani [CCKR11] showed a $O(1/2 - \epsilon)$-approximation for the Rap on the line, which is the special case of Ufp on a line for which all capacities are equal. Subsequently, Chakrabarti et al. [CCGK07] give a $O(1)$-approximation under the Nba. Chekuri, Mydlarz and Shephard [CMS07] showed the first $O(1)$ approximation to Ufp on trees also under the Nba, and also improving on the constant for Ufp on paths obtained in [CCGK07]. A constant factor for UfpPath without the Nba was given by Bonsma et al. [BSW11], and the constant factor was improved to $(2 + \varepsilon)$ by Anagnostopoulos et al. [AGLW14]. For Ufp on trees without the Nba, Chekuri, Ene, and Korula [CEK09] showed a $O(\log^2 p)$ approximation. A linear programming relaxation for the Ufp on trees with an integrality gap of $O(\log p \cdot \min\{\log p, \log n\})$ was shown bu Friggstad and Gao [FG15]. In our work, we will build upon ideas from some of the aforementioned research [CCKR11, CMS07, CCGK07].

**Stochastic Packing Problems.** There has also been extensive research relating to stochastic optimization

problems, tracing all the way back to the work of Dantzig in the 1950s, but the work on approximation algorithms for stochastic problems is more recent. The work of Dean, Goemans and Vondrak [DGV08, Dea05] is closest to our work. They initiated the study of stochastic packing problems: the paper [DGV08] showed an $O(1)$ approximation to Stochastic Knapsack problem, which is the single-link network special case of the sUfp. The linear programming relaxation that we use in our work is a direct extension of one used for the Stochastic Knapsack problem [DGV08].

The sUfp has been studied before: Chawla and Roughgarden [CR06] showed approximation results for the single-source version of the sUfp, i.e., for the case where all demands have a common source. They give approximation algorithms under a stronger variant of the Nba. Before our results, there were no known approximation results for the sUfp on paths or trees, or for extensions of the sUfp where each resource demand may involve multiple nodes.

## 1.4 Our Contributions

We give the first polytime constant-factor approximation algorithms for the sUfpTree-$k$Spider under the Nba and the sRapTree-$k$Spider. The last two results also imply, as a corollary, constant-factor approximation algorithms for the sUfpPath under the Nba, the sUfpTree under the Nba, the sRapPath and the sRapTree. Subsequently, we will however come up with better constant-factor guarantees for the sUfpPath under the Nba, the sUfpTree under the Nba and the sRapPath.

For the sRap$k$Tree-Subtree, we provide a polytime non-adaptive approximation algorithm which achieves a constant-factor approximation to non-adaptive algorithms. Whether this algorithm is a constant-factor approximation to any algorithm for the sRap$k$Tree-Subtree is not known to us and is open to resolution.

In obtaining a constant-factor approximation algorithm for the sUfpTree-$k$Spider, we will first derive a constant-factor approximation algorithm for instances of the UfpTree-$k$Spider where all demands are 1 and capacities are integral, and refer to an argument by means of which this result can be extended to the UfpTree-$k$Spider under the Nba with integral capacities and integral demands.

All algorithms we present are non-adaptive and hence these results imply that the adaptivity gaps for the sRapPath, the sUfpPath under the Nba, the sUfpTree under the Nba and the sRapTree are all constant whereas those for the sUfpTree-$k$Spider and the sRapTree-$k$Spider are polynomial in $k$.

## 1.5 Additional Specifications

In this section we introduce some additional notation and conventions for convenience of the analysis in subsequent chapters.

### 1.5.1 Notation

The *bottleneck capacity of task* $t$ is defined as $b_t := \min\{c_e \mid e \in P_t\}$. The *truncated size* of task $t$ is $\widetilde{S}_t := \min\{S_t, c_{\min}\}$ and its *mean truncated size* is $\mu_t := \mathbf{E}[\widetilde{S}_t]$. The *effective payoff* of task $t$ is defined as $\widetilde{v}_t := v_t \cdot \Pr[S_t \leq b_t]$. Note that under the Nba, $\widetilde{S}_t = S_t$ and $\widetilde{v}_t = v_t$ for all tasks $t$ and that the scaling specified in §1.2 ensures that $c_{\min} = 1$ for all problems we will deal with in our work.

If $A$ denotes a set of tasks, then the set of tasks from $A$ incident on edge $e$ is defined as $A_e := \{t \in A \mid P_t \ni e\}$. We define the truncated size of $A$ as $\mu(A) := \sum_{t \in A} \mu_t$.

### 1.5.2 Feasible Tasks

Since the sizes of the tasks are positive random variables it is not beneficial to schedule a yet-unscheduled task $t$ if there exists a edge in $P_t$ which is completely used up. We call such tasks infeasible, and the remaining yet-unscheduled tasks feasible at this time. For convenience of analysis, we adopt the convention that at a given point in time an algorithm can only schedule tasks feasible at that point in time.

The state at any point in time is completely defined by the residual capacities of all edges and the set of unscheduled tasks at that point in time. Each scheduling algorithm is completely specified by the feasible task it chooses to schedule for every state that may arise as a consequence of its previous decisions. Note that

these choices may not be deterministic. [4] Thus a scheduling algorithm provides a probability distribution over feasible tasks for each state that may arise as a consequence of its previous decisions.

## 1.6 Adaptivity Gap

Algorithms which make decisions based on the outcomes of previously scheduled tasks are referred to as *adaptive algorithms*. Those which premeditate a sequence of distinct tasks and schedule them sequentially, skipping only past the ones which become infeasible in the process are *non-adaptive algorithms*.

Let $\mathbb{A}$ denote the set of all scheduling algorithms for a given instance of the sUfpTree-Subtree. We denote by $\mathbb{A}_N$ the set of all non-adaptive scheduling algorithms for that instance of the sUfpTree-Subtree. Note that $\mathbb{A}_N \subseteq \mathbb{A}$. Let $\mathcal{P}(\mathcal{A})$ denote the random variable which equals the payoff of some scheduling algorithm $\mathcal{A} \in \mathbb{A}$. We define optimal payoff $OPT := \sup\{\mathbf{E}[\mathcal{P}(\mathcal{A})] \mid \mathcal{A} \in \mathbb{A}\}$ and non-adaptive optimal payoff $OPT_N := \sup\{\mathbf{E}[\mathcal{P}(\mathcal{A})] \mid \mathcal{A} \in \mathbb{A}_N\}$. The adaptivity gap for the given problem is the supremum of $OPT/OPT_N$ over all problem instances.

## 1.7 Structure of the Dissertation

We say that task $t \in T$ is $\delta$-small if $\mu_t \leq \delta b_t$ and $\delta$-large if $\mu_t > \delta b_t$ for $\delta \in (0, 1)$. Instances of the sUfpTree-Subtree in which all tasks are $\delta$-small are called $\delta$-small instances and those in which all tasks are $\delta$-large are called $\delta$-large instances. For each special case of the sUfpTree-Subtree relevant to our work we will give separate polytime non-adaptive constant-factor approximation algorithms for $\delta$-small instances and $\delta$-large instances. We will then combine these suitably to provide an algorithm which provides a polytime non-adaptive constant-factor approximation for the problem in consideration.

- In §2 we define a linear program and show that an optimal solution to it is an upper bound on $OPT$. This result holds for sUfpTree-Subtree and is essential to all approximation algorithms in our work.
- In §3 we provide the strategy for combining the approximation algorithms for the $\delta$-small instances and $\delta$-large instances.
- In §4 we describe approximation algorithms for $\delta$-small instances.
- In §5 we describe approximation algorithms for $\delta$-large instances.
- In §6 we apply the results of §3, §4 and §5 to obtain the results that §1.4 outlines.

[4] Derandomization leads to the observation that it is enough to consider algorithms which make deterministic choices at every stage. This observation is however, not useful to our arguments.

# Chapter 2

# A Linear Programming Relaxation

As described in the previous section, the theme of this paper is polytime algorithms, whose expected value is within a constant of the optimal solution to a linear program which is an upper bound for $OPT$. Since in many cases, $OPT$ may itself be significantly smaller than the optimal solution to the linear program, the algorithms may provide significantly better approximations in practice than the constant factors that they guarantee.

This section establishes the LP relaxation which is central to all the results in this paper. The LP and its proof are straightforward extensions of the LP relaxation idea by Dean, Goemans and Vondrak [DGV08]. The relaxation holds for the sUfpTree-Subtree, of which all problems we will consider in this paper are special cases.

**Lemma 2.1** *Fix an algorithm $\mathcal{A}$, and let $A$ denote the set of tasks scheduled by it. For edge $e \in E$, recall that $A_e$ is the set of tasks from $A$ incident on edge $e$. Then*

$$\mathbf{E}[\mu(A_e)] \leq c_e + c_{\min}$$

*(As mentioned in §1.5, $c_{\min} = 1$ for all problem settings that we consider in this paper.)*

**Proof.** Let $A^i$ denote the set of the first $i$ tasks that $\mathcal{A}$ schedules. If $\mathcal{A}$ stops scheduling tasks after scheduling the $i'^{\text{th}}$ task then for $i > i'$ we define $A^i = A^{i'}$. Recall from §1.5 that $A_e$ denotes the set of tasks in $A$ incident on $e$. Note that

$$\mathbf{E}[\mu(A_e)] = \lim_{i \to \infty} \mathbf{E}[\mu(A_e^i)] = \sup_{i \geq 0} \mathbf{E}[\mu(A_e^i)]$$

Recall again from §1.5 that $\widetilde{S}_t = \min\{S_t, c_{\min}\}$. Since $\widetilde{S}_t \leq c_{\min}$ for all tasks $t$ and since $\mathcal{A}$ cannot schedule any task which uses edge $e$ after the size of scheduled tasks which use edge $e$ equals or exceeds $c_e$ we infer that

$$\sum_{t \in A_e^i} \widetilde{S}_t \leq c_e + c_{\min}$$

Define $X_e^i = \sum_{t \in A_e^i} (\widetilde{S}_t - \mu_t)$. Let us observe that $\mathbf{E}[X_e^{i+1} \mid X_e^i] = X_e^i$. If there are no more tasks which $\mathcal{A}$ schedules or if next task to be scheduled does not use edge $e$ then this holds trivially. Otherwise, note that conditioned on the next task to be scheduled $t_{\text{next}}$ we have $\mathbf{E}[X_e^{i+1} \mid X_e^i, t_{\text{next}}] = X_e^i + \mathbf{E}[\widetilde{S}_{t_{\text{next}}}] - \mu_{t_{\text{next}}} = X_e^i$. Thus we can remove the conditioning to once again obtain $\mathbf{E}[X_e^{i+1} \mid X_e^i] = X_e^i$. We have now shown that the sequence $X_e^i$ is a martingale and by the martingale property we conclude that $\mathbf{E}[X_e^i] = \mathbf{E}[X_e^0] = 0$. Linearity of expectation now gives us the result

$$\mathbf{E}[\mu(A_e^i)] = \mathbf{E}\big[\sum_{t \in A_e^i} \widetilde{S}_t\big] \leq c_e + c_{\min}$$

This completes the proof of Lemma 2.1. ∎

## 2.1 The LP relaxations

We now define two LPs that we will use to derive all our approximation results. The first of these, for a particular vector of parameters, is an upper bound on $OPT$ for a given sUfpTree-Subtree instance, as will be shown in Theorem 2.2.

For a vector $\mathbf{u} = (u_1, u_2, \ldots, u_m)$ of "capacities", the following LP is denoted by $\mathcal{L}(\mathbf{u})$:

$$\begin{aligned} & \max \sum_{t \in T} \widetilde{v}_t x_t & & \\ \text{subject to} \quad & \sum_{t \in T_e} \mu_t x_t \leq u_e & \forall e \in E & \quad (L.1) \\ & 0 \leq x_t \leq 1 & \forall t \in T & \quad (L.2) \end{aligned}$$

Let $\phi(\mathbf{u})$ be the value of the optimal solution to $\mathcal{L}(\mathbf{u})$.

We define a second LP which will be useful in obtaining approximation results for $\delta$-large instances. For the same set of parameters, this second LP differs from the first only in that it replaces the $\mu_t$s in the constraints with 1s. Since we know that from the Nba that $\mu_t \leq 1$ we know that the constraints of the second LP are stronger than those of the first. We will go on to show that for $\delta$-large tasks, the LPs are within a $O(1/\delta)$ factor of each other.

For a vector $\mathbf{u} = (u_1, u_2, \ldots, u_m)$ of "capacities", the following LP is denoted by $\mathcal{L}'(\mathbf{u})$:

$$\begin{aligned} & \max \sum_{t \in T} \widetilde{v}_t x_t & & \\ \text{subject to} \quad & \sum_{t \in T_e} x_t \leq u_e & \forall e \in E & \quad (L'.1) \\ & 0 \leq x_t \leq 1 & \forall t \in T & \quad (L'.2) \end{aligned}$$

Let $\phi'(\mathbf{u})$ denote the value of the optimal solution to $\mathcal{L}'(\mathbf{u})$.

**Theorem 2.2** *The expected payoff of every scheduling algorithm for an instance of the sUfpTree-Subtree is bounded above by $\phi(\mathbf{c} + \mathbf{c}_{\min})$.* [1] *Hence*

$$OPT \leq \phi(\mathbf{c} + \mathbf{c}_{\min})$$

**Proof.** Consider a scheduling algorithm $\mathcal{A}$ and let $A$ denote the set of tasks that it schedules. Let $\mathbf{x}$ denote the vector of size $n$ whose $t^{\text{th}}$ component $x_t = \Pr[t \in A]$. From this definition and from Lemma 2.1 we have that for all edges $e$

$$\sum_{t \in A_e} \mu_t x_t = \mathbf{E}[\mu(A_e)] \leq c_e + c_{\min}$$

We infer that the $x_t$'s satisfy the set of constraints (L.1). That they satisfy the set of constraints (L.2) follows trivially from their definition. Hence

$$\sum_{t \in T} \widetilde{v}_t x_t \leq \phi(\mathbf{c} + \mathbf{c}_{\min})$$

Let $G_t$ denote the (random) indicator variable which indicates whether task $t$ is successfully scheduled given that it was scheduled by $\mathcal{A}$. Observe that $G_t = 1$ implies that $S_t \leq b_t$. Hence

$$\begin{aligned} \text{Expected profit due to task } t &= \mathbf{E}[v_t e_t \mid t \in A] \Pr[t \in A] \\ &= v_t \Pr[G_t = 1 \mid t \in A] \Pr[t \in A] \\ &\leq v_t \Pr[S_t \leq b_t] \Pr[t \in A] \\ &= \widetilde{v}_t x_t \end{aligned}$$

Hence $\mathcal{P}(\mathcal{A}) \leq \sum_{t \in T} \widetilde{v}_t x_t \leq \phi(\mathbf{c} + \mathbf{c}_{\min})$ and since this is true for any $\mathcal{A} \in \mathbb{A}$ we conclude that $OPT \leq \phi(\mathbf{c} + \mathbf{c}_{\min})$. ∎

[1] Note that for every scalar $s$, we use $\mathbf{s}$ to denote the vector of dimension $m$ having all entries equal to $s$.

# Chapter 3

# Combining Approximation Algorithms

The idea of separately tackling and then combining approaches for $\delta$-large and $\delta$-small instances has often been used in addressing the Ufp and its special cases [CCKR11, CCGK07].

**Theorem 3.1** *Consider an instance $\mathcal{I} = (G, \mathbf{c}, T)$ of the sUfpTree-Subtree with optimal payoff $OPT$ specified by the graph $G$, edge-capacity vector $\mathbf{c}$ and the set of tasks $T$. Let $T_1$ and $T_2$ form a partition of $T$ and consider instances $\mathcal{I}_1 = (G, \mathbf{c}, T_1)$ and $\mathcal{I}_2 = (G, \mathbf{c}, T_2)$ with optimal payoffs $OPT_1$ and $OPT_2$.*

*If there exists, for instance $\mathcal{I}_1$, a polytime non-adaptive algorithm $\mathcal{A}_1$, a polytime computable quantity $\xi_1$ and a constant $\alpha_1 \geq 1$ such that,*

$$\mathbf{E}[\mathcal{P}(\mathcal{A}_1)] \geq \xi_1 \geq \frac{1}{\alpha_1} OPT_1$$

*and there exists, for instance $\mathcal{I}_2$, a polytime non-adaptive algorithm $\mathcal{A}_2$, a polytime computable quantity $\xi_2$ and a constant $\alpha_2 \geq 1$ such that,*

$$\mathbf{E}[\mathcal{P}(\mathcal{A}_2)] \geq \xi_2 \geq \frac{1}{\alpha_2} OPT_2$$

*then there exists, for instance $\mathcal{I}$ a polytime non-adaptive algorithm $\mathcal{A}$ such that,*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{\alpha_1 + \alpha_2} OPT$$

**Proof.** Consider an optimal algorithm $\mathcal{A}_{OPT}$ for instance $\mathcal{I}$, which has expected payoff $OPT$. Now consider an algorithm $\mathcal{A}'_1$ for $\mathcal{I}_1$ which makes the same decisions as $\mathcal{A}_{OPT}$, instantiating but not scheduling the tasks in $T_2$ that $\mathcal{A}_{OPT}$ would have scheduled. Since the residual capacity of each edge in $G$ at each stage under $\mathcal{A}'_1$ is at least that under $\mathcal{A}_{OPT}$, we infer that the payoff of $\mathcal{A}'_1$ is at least the payoff of $\mathcal{A}_{OPT}$ due to tasks in $T_1$. Similarly there exists an algorithm $\mathcal{A}'_2$ for $\mathcal{I}_2$ whose payoff is at least the payoff of $\mathcal{A}_{OPT}$ due to tasks in $T_2$. We conclude that

$$\begin{aligned} OPT &\leq OPT_1 + OPT_2 \\ &\leq \alpha_1 \xi_1 + \alpha_2 \xi_2 \\ \max(\xi_1, \xi_2) &\geq \frac{OPT}{\alpha_1 + \alpha_2} \end{aligned}$$

The algorithm $\mathcal{A}$ does the following: It computes $\xi_1$ and $\xi_2$ in polytime. If $\xi_1$ is the greater of the two it executes $\mathcal{A}_1$ and ignores the tasks in $T_2$, otherwise it executes $\mathcal{A}_2$ and ignores the tasks in $T_1$. We conclude that

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \max(\xi_1, \xi_2) \geq \frac{1}{\alpha_1 + \alpha_2} OPT$$

Finally, we observe that from the definition of $\mathcal{A}$ that it is a polytime non-adaptive algorithm. This completes the proof of the claim. ∎

We will use Theorem 3.1 to obtain approximation algorithms for the sRapPath, the sUfpPath under the Nba, the sRapTree, the sUfpTree under the Nba, the sRapTree-$k$Spider and the sUfpTree-$k$Spider under the Nba. To obtain an algorithm for sRap$k$Tree-Subtree which is a constant-factor approximation to non-adaptive algorithms, we need a slight variant of Theorem 3.1, stated below. Its proof is nearly identical and is not stated here so as to avoid repetition.

**Theorem 3.2** *Consider an instance $\mathcal{I} = (G, \mathbf{c}, T)$ of the sUfpTree-Subtree with non-adaptive optimal payoff $OPT_N$ specified by the graph $G$, edge-capacity vector $\mathbf{c}$ and the set of tasks $T$. Let $T_1$ and $T_2$ form a partition of $T$ and consider instances $\mathcal{I}_1 = (G, \mathbf{c}, T_1)$ and $\mathcal{I}_2 = (G, \mathbf{c}, T_2)$ with non-adaptive optimal payoffs $OPT_{N1}$ and $OPT_{N2}$.*

*If there exists, for instance $\mathcal{I}_1$, a polytime non-adaptive algorithm $\mathcal{A}_1$, a polytime computable quantity $\xi_1$ and a constant $\alpha_1 \geq 1$ such that,*

$$\mathbf{E}[\mathcal{P}(\mathcal{A}_1)] \geq \xi_1 \geq \frac{1}{\alpha_1} OPT_{N1}$$

*and there exists, for instance $\mathcal{I}_2$, a polytime non-adaptive algorithm $\mathcal{A}_2$, a polytime computable quantity $\xi_2$ and a constant $\alpha_2 \geq 1$ such that,*

$$\mathbf{E}[\mathcal{P}(\mathcal{A}_2)] \geq \xi_2 \geq \frac{1}{\alpha_2} OPT_{N2}$$

*then there exists, for instance $\mathcal{I}$ a polytime non-adaptive algorithm $\mathcal{A}$ such that,*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{\alpha_1 + \alpha_2} OPT_N$$

# Chapter 4

# Algorithms for Small Tasks

In this section we give approximation algorithms for $\delta$-small instances of various special cases of the sUfpTree-Subtree. As defined in §1.7, we say that a task $t$ is $\delta$-small if $\mu_t \leq \delta b_t$ and an instance is $\delta$-small if all tasks in it are $\delta$-small.

## 4.1 Uniform Capacities

When all capacities are equal (and scaled to 1), a task $t$ is $\delta$-small if $\mu_t \leq \delta$. The following theorem gives an approximation for $\delta$-small instances of the sRapTree-$k$Subtree. We will use this result to obtain constant-factor approximations for the sRapPath and sRapTree-$k$Spider and to obtain a constant-factor approximation to non-adaptive algorithms for sRap$k$Tree-Subtree. This theorem is an extension of the randomized rounding approach for dealing with $\delta$-small instances of the RapPath devised by Calinescu, Chakrabarti, Karloff and Rabani. [CCKR11]

**Theorem 4.1** *Consider the sRapTree-$k$Subtree. For reals $q, \delta \in (0,1)$ such that $\delta < q$, there exists a polytime non-adaptive scheduling algorithm $\mathcal{A}(\delta)$ which for $\delta$-small instances guarantees*

$$\mathbf{E}[\mathcal{P}(\mathcal{A}(\delta))] \geq \frac{(1-q)(1-\frac{1}{q}\delta)}{8k} \cdot \phi(\mathbf{2}) \geq \frac{(1-q)(1-\frac{1}{q}\delta)}{8k} \cdot OPT$$

**Proof.** We will use randomized rounding on an optimal solution to $\phi(\mathbf{2})$ to obtain a constant-factor approximation. Let $(x_1, \ldots, x_n)$ be a solution to $\mathcal{L}(\mathbf{2})$ for which the value of the objective function $\sum_{t \in T} \widetilde{v}_t x_t$ equals $\phi(\mathbf{2})$. The idea is to schedule tasks $t$ with probability proportional to $x_t$, with the constant of proportionality being sufficiently small to ensure that when each task is scheduled, there is at least a constant probability that all of its edges will have residual capacity at least a constant multiple of $\delta$.

We define the sequence $Y$ of random indicator variables, one for each task, as follows:

$$Y_t = \begin{cases} 1 & \text{with probablility } \frac{(1-\frac{1}{q}\delta)}{4k} x_t \\ 0 & \text{otherwise} \end{cases}$$

*Algorithm:* The algorithm $\mathcal{A}(\delta)$ proceeds as follows: It considers the tasks in the increasing order of their indices and attempts to schedule task $t$ if $Y_t = 1$ and the task $t$ is feasible at that point of time. Recall that the tasks are numbered in increasing order of the depth of the root of the subtree corresponding to them.

We now define another sequence of random indicator variables $Z$, one for each task. $Z_t$ equals 1 if the algorithm $\mathcal{A}$ decides to schedule task $t$ (equivalently $Y_t = 1$) and there is enough space (at least $1/q \cdot \mu_t$) on each edge of $P_t$ at the time of scheduling task $t$.

$$Z_t = \begin{cases} 1 & \text{if } Y_t = 1 \text{ and } \sum_{\substack{t' < t \\ t' \in T_e}} S_{t'} Z_{t'} \leq 1 - \frac{1}{q}\mu_t \text{ for all } e \in P_t \\ 0 & \text{otherwise} \end{cases}$$

We will prove the following claims:

**Claim 4.2** $\Pr[Z_t = 1 \mid Y_t = 1] \geq \frac{1}{2}$

**Claim 4.3** $\Pr[\textit{Task } t \textit{ is successfully scheduled} \mid Z_t = 1] \geq 1 - q$

Assuming that these claims are true, we can complete the proof as follows:

$$\begin{aligned}\Pr[\text{Task } t \text{ is successfully scheduled}] &= \Pr[Y_t = 1] \cdot \Pr[Z_t = 1 \mid Y_t = 1] \\ &\quad \cdot \Pr[\text{Task } t \text{ is successfully scheduled} \mid Z_t = 1, Y_t = 1] \\ &= \Pr[Y_t = 1] \cdot \Pr[Z_t = 1 \mid Y_t = 1] \\ &\quad \cdot \Pr[\text{Task } t \text{ is successfully scheduled} \mid Z_t = 1] \\ &\geq \frac{1 - \frac{1}{q}\delta}{4k} x_t \cdot \frac{1}{2} \cdot (1 - q) = \frac{(1-q)(1-\frac{1}{q}\delta)}{8k} x_t\end{aligned}$$

The expected payoff from task $t$ equals $v_t \Pr[\text{Task } t \text{ is successfully scheduled}]$. Using Theorem 2.2, we infer that

$$\begin{aligned}\mathbf{E}[\mathcal{P}(\mathcal{A})] &= \sum_{t \in T} v_t \Pr[\text{Task } t \text{ is successfully scheduled}] \\ &\geq \frac{(1-q)(1-\frac{1}{q}\delta)}{8k} \sum_{t \in T} v_t x_t \geq \frac{(1-q)(1-\frac{1}{q}\delta)}{8k} \sum_{t \in T} \widetilde{v}_t x_t \\ &= \frac{(1-q)(1-\frac{1}{q}\delta)}{8k} \phi(\mathbf{2}) \geq \frac{(1-q)(1-\frac{1}{q}\delta)}{8k} OPT\end{aligned}$$

This completes the proof of Theorem 4.1. ∎

Let us now prove the Claims 4.2 and 4.3.

**Claim 4.2** $\Pr[Z_t = 1 \mid Y_t = 1] \geq \frac{1}{2}$

**Proof.** Recall the definition of the r.v.s $Y_t$ and $Z_t$ in the proof of Theorem 4.1. $Z_t$ equals 1 if the algorithm $\mathcal{A}$ decides to schedule task $t$ (equivalently $Y_t = 1$) and there is enough space (at least $1/q \cdot \mu_t$) on each edge of $P_t$ at the time of scheduling task $t$. The basic idea underlying this proof is that is that since we schedule tasks in increasing order of depth and since all edges have equal capacities, if there is enough space on the least-depth edges of $P_t$ (which are at most $k$ in number), then there is enough space on all edges of $P_t$.

We denote by $\mathcal{Y}(t, e)$ an expression which is an upper bound on the used up capacity on edge $e$ at the stage where decisions on the first $t-1$ tasks have been made by $\mathcal{A}$. We also introduce another analogous piece of notation $\widetilde{\mathcal{Y}}(t, e)$. For $t \in T$ and $e \in E$ define

$$\mathcal{Y}(t, e) = \sum_{\substack{t' < t \\ t' \in T_e}} S_{t'} Y_{t'} \qquad \widetilde{\mathcal{Y}}(t, e) = \sum_{\substack{t' < t \\ t' \in T_e}} \widetilde{S}_{t'} Y_{t'}$$

If $Y_{t'} = 1$ and $t'$ was feasible at the point in time when the first $t' - 1$ decisions were made then $S_{t'}$ is well-defined since we know that an attempt was made to schedule task $t'$. We adopt the convention theat if $Y_{t'} = 1$ and $t'$ was not feasible at the point in time when the first $t' - 1$ decisions were made then it was instantiated even though no attempt was made to insert it. Under this convention the expressions $\mathcal{Y}(t, e)$ and $\widetilde{\mathcal{Y}}(t, e)$ are always well-defined.

Consider $t \in T$. Let $x$ denote the root of $P_t$ and let $f_1, \ldots, f_{k'}$ denote the least-depth edges between $x$ and each of its $k' (\leq k)$ children in $P_t$. Since we are processing tasks in top-down order of the roots of their corresponding subtrees and since all edges have equal capacities, one of $f_1, \ldots, f_{k'}$ must have minimum

residual capacity among edges in $P_t$. We conclude that if $Z_t = 0$ and $Y_t = 1$ then there must exist an $i \in [1, k']$ such that

$$\mathcal{Y}(t, f_i) \geq \sum_{\substack{t'<t \\ t' \in T_{f_i}}} S_{t'} Z_{t'} > 1 - \frac{1}{q}\mu_t$$

From the union bound, we have

$$\Pr[Z_t = 0 \mid Y_t = 1] \leq \sum_{i=1}^{k'} \Pr[\mathcal{Y}(t, f_i) > 1 - \frac{1}{q}\mu_t]$$

Consider a particular $\mathcal{Y}(t, f_i)$ summation. If none of the $S_{t'}$ values which contribute to it are more than one then we can replace all of them by $\widetilde{S}_{t'}$. If any of them is more than one then the inequality is guaranteed to be satisfied and again we can replace all $S_{t'}$ values with $\widetilde{S}_{t'}$ values. Hence we can replace each $\mathcal{Y}(t, f_i)$ with $\widetilde{\mathcal{Y}}(t, f_i)$. We can also replace the truncated means with $\delta$ since we are dealing with $\delta$-small tasks.

$$\Pr[Z_t = 0 \mid Y_t = 1] \leq \sum_{i=1}^{k'} \Pr[\widetilde{\mathcal{Y}}(t, f_i) > 1 - \frac{1}{q}\mu_t] \leq \sum_{i=1}^{k'} \Pr[\widetilde{\mathcal{Y}}(t, f_i) > 1 - \frac{1}{q}\delta]$$

The $Y_{t'}$ variables are independent from the $\widetilde{S}_{t'}$ variables. Hence

$$\begin{aligned}
\mathbf{E}[\widetilde{\mathcal{Y}}(t, f_i)] &= \mathbf{E}[\sum_{\substack{t'<t \\ t' \in T_{f_i}}} \widetilde{S}_{t'} Y_{t'}] = \sum_{\substack{t'<t \\ t' \in T_{f_i}}} \mathbf{E}[\widetilde{S}_{t'}]\mathbf{E}[Y_{t'}] = \sum_{\substack{t'<t \\ t' \in T_{f_i}}} \mu_{t'} \cdot \frac{1 - \frac{1}{q}\delta}{4k} x_{t'} \\
&= \frac{1 - \frac{1}{q}\delta}{2k} \cdot \sum_{\substack{t'<t \\ t' \in T_{f_i}}} \frac{\mu_{t'} x_{t'}}{2} \leq \frac{1 - \frac{1}{q}\delta}{2k} \cdot \sum_{t' \in T_{f_i}} \frac{\mu_{t'} x_{t'}}{2} \leq \frac{1 - \frac{1}{q}\delta}{2k}
\end{aligned}$$

The last of the inequalities above follows from the set of constraints (L.1). We now apply Markov's inequality to infer that

$$\Pr[\widetilde{\mathcal{Y}}(t, f_i) > 1 - \frac{1}{q}\delta] \leq \frac{1}{2k}$$

Consequently

$$\Pr[Z_t = 0 \mid Y_t = 1] \leq \sum_{i=1}^{k'} \Pr[\widetilde{\mathcal{Y}}(t, f_i) > 1 - \frac{1}{q}\delta] \leq \frac{k'}{2k} \leq \frac{1}{2}$$

Hence

$$\Pr[Z_t = 1 \mid Y_t = 1] \geq \frac{1}{2}$$

This completes the proof of Claim 4.2. ∎

**Claim 4.3** $\Pr[\text{Task } t \text{ is successfully scheduled} \mid Z_t = 1] \geq 1 - q$

**Proof.** Consider the point in time when we have made the first $t - 1$ decisions. Note that $Z_t = 1$ implies that $Y_t = 1$. It also implies that the used up capacity of every edge in $P_t$ is at most $(1 - 1/q \cdot \mu_t)$. The first consequence of this is that task $t$ is feasible and an attempt will be made by $\mathcal{A}(\delta)$ to schedule it. The Markov inequality implies that $\Pr[\widetilde{S}_t > 1/q \cdot \mu_t] < q$ or equivalently $\Pr[\widetilde{S}_t \leq 1/q \cdot \mu_t] \geq 1 - q$. Since $\mu_t \leq \delta < q$ we know that $\widetilde{S}_t \leq 1/q \cdot \mu_t < 1 \Rightarrow \widetilde{S}_t = S_t$ from which we conclude that $\Pr[S_t \leq 1/q \cdot \mu_t] \geq 1 - q$. This leads us to the second more important consequence:

$$\Pr[\text{Task } t \text{ is successfully scheduled} \mid Z_t = 1] \geq 1 - q$$

This completes the proof of Claim 4.3. ∎

## 4.2 Non-Uniform Capacities

The following theorem gives an approximation for $\delta$-small instances of the sUfpTree-$k$Spider under the Nba. The algorithm is an adaptation of the approach for dealing with small tasks for the Ufp on a path under the Nba devised by Chakrabarti, Chekuri, Gupta and Kumar [CCGK07]. We will use this result to obtain constant-factor approximations for the sUfpTree-$k$Spider under the Nba. We can also use it to obtain constant-factor approximations under the Nba for the sUfpPath and the sUfpTree, since they are special cases of sUfpTree-$k$Spider. However we will use better results (described later in this section) for these in order to attain a better constant-factor.

**Theorem 4.4** *Consider $\delta = 0.0005$. For $\delta$-small instances of the sUfpTree-$k$Spider under the Nba there exists a non-adaptive scheduling algorithm $\mathcal{A}$ for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{822.37 \cdot k^{2.15}} \cdot \phi(\mathbf{c}+\mathbf{1}) \geq \frac{1}{822.37 \cdot k^{2.15}} \cdot OPT$$

**Proof.** The general structure of this proof is similar the proof of Theorem 4.1. As we did previously, we use randomized rounding on an optimal solution to $\phi(\mathbf{c}+\mathbf{1})$ to obtain a constant-factor approximation. However, in this setting it is harder to show that the probability that a task is successfully scheduled that given that it is scheduled is at least a constant. This is because the unequal capacities imply that it no longer suffices to just consider the topmost edges of the *k-spider* corresponding to each task. The workaround to this, an adaptation of the idea in [CCGK07], is delineated in the proof of Claim 4.5.

Let $(x_1, \ldots, x_n)$ be a solution to $\mathcal{L}(\mathbf{c}+\mathbf{1})$ for which the value of the objective function $\sum_{t \in T} \widetilde{v}_t x_t$ equals $\phi(\mathbf{c}+\mathbf{1})$. As in the proof of Theorem 4.7, the idea is to schedule tasks $t$ with probability proportional to $x_t$, with the constant of proportionality being sufficiently small to ensure that when each task is scheduled, there is at least a constant probability that all of its edges will have residual capacity at least a constant multiple of $\delta b_t$.

We define a parameter $\alpha = 0.032/k^{2.15}$ and the sequence $Y$ of random indicator variables, one for each item, as follows:

$$Y_t = \begin{cases} 1 & \text{with probablility } \frac{\alpha x_t}{2} \\ 0 & \text{otherwise} \end{cases}$$

*Algorithm:* The algorithm $\mathcal{A}$ proceeds as follows: It considers the tasks in the increasing order of their indices and attempts to schedule task $t$ if $Y_t = 1$ and task $t$ is feasible at that point of time. Recall that in this case the tasks are numbered in increasing depth of the root of the *k-spiders* corresponding to them.

We now define another sequence of random indicator variables $Z$, one for each task. $Z_t$ equals 1 if the algorithm $\mathcal{A}$ decides to schedule task $t$ (equivalently $Y_t = 1$) and there is enough space (at least $2 \cdot \mu_t$) on each edge of $P_t$ at the time of scheduling task $t$.

$$Z_t = \begin{cases} 1 & \text{if } Y_t = 1 \text{ and } \sum\limits_{\substack{t' < t \\ t' \in T_e}} S_{t'} Z_{t'} \leq c_e - 2\mu_t \text{ for all } e \in P_t \\ 0 & \text{otherwise} \end{cases}$$

We will prove the following claims:

**Claim 4.5** $\Pr[Z_t = 1 \mid Y_t = 1] \geq 0.154$

**Claim 4.6** $\Pr[\textit{Task } t \textit{ is successfully scheduled} \mid Z_t = 1] \geq \frac{1}{2}$

Assuming that these claims are true, we can complete the proof as follows:

$$\begin{aligned}\Pr[\text{Task } t \text{ is successfully scheduled}] &= \Pr[Y_t = 1] \cdot \Pr[Z_t = 1 \mid Y_t = 1] \\ &\cdot \Pr[\text{Task } t \text{ is successfully scheduled} \mid Z_t = 1, Y_t = 1] \\ &= \Pr[Y_t = 1] \cdot \Pr[Z_t = 1 \mid Y_t = 1] \\ &\cdot \Pr[\text{Task } t \text{ is successfully scheduled} \mid Z_t = 1] \\ &\geq \frac{\alpha x_t}{2} \cdot 0.154 \cdot \frac{1}{2} \\ &\geq 0.038\alpha \cdot x_t\end{aligned}$$

Thus the expected payoff from task $t$ is at least $0.038\alpha \cdot v_t x_t$. Hence

$$\begin{aligned}\mathbf{E}[\mathcal{P}(\mathcal{A})] &= \sum_{t \in T} v_t \Pr[\text{Task } t \text{ is successfully scheduled}] \\ &\geq 0.038\alpha \cdot \sum_{t \in T} v_t x_t = 0.038\alpha \cdot \phi(\mathbf{c} + \mathbf{1}) \\ &\geq 0.038\alpha \cdot OPT \geq \frac{1}{822.37 \cdot k^{2.15}} \cdot OPT\end{aligned}$$

This completes the proof of Theorem 4.4. ∎

Let us now prove Claims 4.5 and 4.6.

**Claim 4.5** $\Pr[Z_t = 1 \mid Y_t = 1] \geq 0.154$

**Proof.** This claim is similar to Claim 4.3. Recall the definition of the r.v.s $Y_t$ and $Z_t$ in the proof of Theorem 4.4. $Z_t$ equals 1 if the algorithm $\mathcal{A}$ decides to schedule task $t$ (equivalently $Y_t = 1$) and there is enough space (at least $2 \cdot \mu_t$) on each edge of $P_t$ at the time of scheduling task $t$. The basic idea underlying this proof is that is that we can extract a sequence of edges with exponentially decreasing capacities (by a factor of at least 2) along each downward path of the *k-spider* $P_t$, such that if any edge in $P_t$ is violated (i.e. there is not enough space on it at the stage that task $t$ is scheduled) then half of one of the extracted edges is violated.

We continue using the $\mathcal{Y}(t, e)$ notation that we introduced in Claim 4.3. The note in Claim 4.3 about instantiation by convention is applicable here as well.

Consider $t \in T$. Let $x$ denote the root of the *k-spider* $P_t$ and let $P_{t,1} = (f_{1,1}, \ldots f_{1,g_1})$, $\ldots P_{t,k'} = (f_{k',1}, \ldots f_{k',g_{k'}})$ be the $k' (\leq k)$ edge-disjoint paths going down from $x$ to the leaves of $P_t$ such that $P_t = P_{t,1} \cup \ldots P_{t,k'}$.

Note that if $Z_t = 0$ and $Y_t = 1$ there exists $i \in [k'], b \in [g_i]$ such that

$$\mathcal{Y}(t, f_{i,b}) \geq \sum_{\substack{t' < t \\ t' \in T_{f_{i,b}}}} S_{t'} Z_{t'} \geq c_{f_{i,b}} - 2\mu_t$$

We will now define a maximal sequence of edges in each $P_{t,i}$ starting at $f_{i,1}$ in which the capacities decrease exponentially. Let us define the sequence $e_{i,1}, \ldots, e_{i,h_i}$ as follows:

$$\begin{aligned}e_{i,1} &= f_{i,1} \\ e_{i,l} &= \arg\min \left\{ \text{depth}(e) \mid e \in P_{t,i} \text{ and } \text{depth}(e_{i,l-1}) < \text{depth}(e) \text{ and } c_e < \frac{c_{e_{i,l-1}}}{2} \right\} \text{ for } l \geq 2\end{aligned}$$

We stop the construction of the sequence when no such $e$ exists. We now define the event $\xi_{i,a}$ for $i \in [k'], a \in [h_i]$ as follows:

$$\xi_{i,a} : \mathcal{Y}(t, e_{i,a}) > \frac{1}{2} c_{e_{i,a}} - 2\mu_t$$

We will now show that $\Pr[Z_t = 0 \mid Y_t = 1] < \sum_{i \in [k']} \sum_{a \in [h_i]} \Pr[\xi_{i,a}]$. We have already noted that if $Z_t = 0$ given that $Y_t = 1$ then we must have $\mathcal{Y}(t, f_{i,b}) > c_{f_{i,b}} - 2\mu_t$ for some $i \in [k'], b \in [g_i]$. Consider some such $i, b$

and let $a \in [h]$ be the largest index such that $depth(e_{i,a}) \leq depth(f_{i,b})$. Note that $a$ is always well-defined because $e_{i,1} = f_{i,1}$. Observe that $\mathcal{Y}(t, e_{i,a}) \geq \mathcal{Y}(t, f_{i,b})$ since we are processing tasks in increasing order of depth and since *k-spiders* which contain $f_{i,1}$ and $f_{i,b}$ must contain $e_{i,a}$. If the inequality $c_{f_{i,b}} < c_{e_{i,a}}/2$ was true then either $f_{i,b}$ or something of a lower depth would have been selected as $e_{i,a+1}$. We thus conclude that $c_{f_{i,b}} \geq c_{e_{i,a}}/2$ and hence that $\mathcal{Y}(t, e_{i,a}) \geq \mathcal{Y}(t, f_{i,b}) > c_{f_{i,b}} - 2\mu_t \geq c_{e_{i,a}}/2 - 2\mu_t$. We infer that if $Z_t = 0$ and $Y_t = 1$ then there exists $i \in [k'], a \in [h_i]$ such that the event $\xi_{i,a}$ occurs. It follows that

$$\Pr[Z_t = 0 \mid Y_t = 1] < \sum_{i \in [k']} \sum_{a \in [h_i]} \Pr[\xi_{i,a}]$$

Let us now upper bound the quantity $\Pr[\xi_{i,a}]$ using the Chernoff bound. We set the parameter $\beta = (1/2 - 2\delta - \alpha)/\alpha$. Observe that

$$\beta = \frac{0.5 - 0.001 - 0.032/k^{2.15}}{0.032/k^{2.15}} \geq \frac{0.467}{0.032} \cdot k^{2.15}$$

Let us define $\beta_0 = (0.467/0.032) \cdot k^{2.15}$, so that $\beta \geq \beta_0$. We use this estimate along with the inequality $\mu_t \leq \delta b_t \leq \delta c_{e_{i,a}}$ to obtain

$$\begin{aligned}\Pr[\xi_{i,a}] &= \Pr[\mathcal{Y}(t, e_{i,a}) > \frac{1}{2} c_{e_{i,a}} - 2\mu_t] \leq \Pr[\mathcal{Y}(t, e_{i,a}) > \frac{1}{2} c_{e_{i,a}} - 2\delta c_{e_{i,a}}] \\ &= \Pr[\mathcal{Y}(t, e_{i,a}) > (1+\beta)\alpha c_{e_{i,a}}] \leq \Pr[\mathcal{Y}(t, e_{i,a}) > (1+\beta_0)\alpha c_{e_{i,a}}]\end{aligned}$$

Since $(x_1, \ldots, x_n)$ is a solution to $\mathcal{L}(\mathbf{c} + \mathbf{1})$, it follows from the capacity constraints (L.1) of $\mathcal{L}(\mathbf{c} + \mathbf{1})$ and the observation $c_{e_{i,a}} \geq c_{\min} = 1$ that

$$\begin{aligned}\mathbf{E}[\mathcal{Y}(t, e_{i,a})] &= \sum_{\substack{t' < t \\ t' \in T_{e_{i,a}}}} \mathbf{E}[S_{t'}]\mathbf{E}[Y_{t'}] = \frac{\alpha}{2} \sum_{\substack{t' < t \\ t' \in T_{e_{i,a}}}} \mu_{t'} x_{t'} \\ &\leq \frac{\alpha}{2} \sum_{t' \in T_{e_{i,a}}} \mu_{t'} x_{t'} \leq \frac{\alpha}{2}(c_{e_{i,a}} + 1) \leq \alpha c_{e_{i,a}}\end{aligned}$$

Note that $\mathcal{Y}(t, e_{i,a})$ is the sum of independent random variables over $[0, 1]$ since we are operating under the Nba and hence we can use the Chernoff bound to obtain that

$$\begin{aligned}\Pr[\xi_{i,a}] &\leq \Pr[\mathcal{Y}(t, e_{i,a}) > (1+\beta_0)\alpha c_{e_{i,a}}] \\ &\leq \Big(\frac{e^{\beta_0}}{(1+\beta_0)^{1+\beta_0}}\Big)^{\alpha c_{e_{i,a}}} \leq \Big(\frac{e}{\beta_0}\Big)^{\beta_0 \alpha c_{e_{i,a}}}\end{aligned}$$

We substitute the values $\beta_0 = (0.467/0.032) \cdot k^{2.15}$ and $\alpha = 0.032/k^{2.15}$

$$\Pr[\xi_{i,a}] \leq \Big(\frac{0.187}{k^{2.15}}\Big)^{0.467 \cdot c_{e_{i,a}}} \leq \Big(\frac{0.458}{k}\Big)^{c_{e_{i,a}}}$$

Finally, we use this result to upper bound $\Pr[Z_t = 0 \mid Y_t = 1]$. We know that $c_{e_{i,a}} > 2c_{e_{i,a+1}}$ for all $a \in [h-1]$ and that $c_{e_{i,h_i}} \geq 1$. Hence

$$\begin{aligned}\Pr[Z_t = 0 \mid Y_t = 1] &\leq \sum_{i \in [k']} \sum_{a=1}^{h_i} \Pr[\xi_{i,a}] \leq \sum_{i \in [k']} \sum_{a=1}^{h_i} \Big(\frac{0.458}{k}\Big)^{c_{e_{i,a}}} \\ &\leq \sum_{i \in [k']} \sum_{j=0}^{\infty} \Big(\frac{0.458}{k}\Big)^{2^j} \leq \sum_{i \in [k']} \sum_{j=0}^{\infty} \Big(\frac{0.458}{k}\Big)^{j} \\ &\leq k' \cdot \Big(\frac{\frac{0.458}{k}}{1 - \frac{0.458}{k}}\Big) \leq k \cdot \Big(\frac{\frac{0.458}{k}}{1 - 0.458}\Big) \leq 0.846\end{aligned}$$

It follows that

$$\Pr[Z_t = 1 \mid Y_t = 1] \geq 0.154$$

This completes the proof of Claim 4.5. ∎

**Claim 4.6** $\Pr[\text{Task } t \text{ is successfully scheduled} \mid Z_t = 1] \geq \frac{1}{2}$

**Proof.** Note that $Z_t = 1$ implies that $Y_t = 1$. It also implies that $\mathcal{Y}(t, e) \leq c_e - 2\mu_t$ for every edge $e \in P_t$. The first consequence of this is that task $t$ is feasible and an attempt will be made by $\mathcal{A}$ to schedule it. The Markov inequality implies that $\Pr[S_t > 2\mu_t] < \frac{1}{2}$ or equivalently $\Pr[S_t \leq 2\mu_t] \geq \frac{1}{2}$. This leads us to the second more important consequence:

$$\Pr[\text{Task } t \text{ is successfully scheduled} \mid Z_t = 1] \geq \frac{1}{2}$$

This completes the proof of Claim 4.6. ∎

### 4.2.1 An Improved Constant for the sUfpPath

Note that the sUfpPath is the special case of sUfpTree-1Spider where $G$ is a path rooted at one of its endpoints. Hence we can obtain a constant-factor approximation for $\delta$-small instances of the sUfpPath using Theorem 4.4. However, in Theorem 4.4 we use several loose upper bounds, including one in which we approximated $\sum_{j\geq 0} z^{2^j}$ by $\sum_{j\geq 0} z^j$. The following theorem gives a better approximation factor to the sUfpPath than the $1/822.37$ factor guaranteed by Theorem 4.4. Its proof is nearly identical to Theorem 4.4 and so only the parts of the calculations which are different are stated here so as to avoid repetition.

**Theorem 4.7** *Consider $\delta = 0.0005$. For $\delta$-small instances of the sUfpPath under the Nba there exists a non-adaptive scheduling algorithm $\mathcal{A}$ for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{310.18} \cdot \phi(\mathbf{c} + \mathbf{1}) \geq \frac{1}{310.18} \cdot OPT$$

**Proof.** As stated above, this proof is nearly identical to that of Theorem 4.4 we will only state here a few calculations which differ from the proof of Theorem 4.4.

As before we define the constants $\alpha = 0.032$ and $\beta = (1/2 - 2\delta - \alpha)/\alpha$. We obtain

$$\Pr[\xi_{1,a}] \leq \left(\frac{e^\beta}{(1+\beta)^{1+\beta}}\right)^{\alpha c_{e_{1,a}}} \leq (0.4051)^{c_{e_{1,a}}}$$

This leads to

$$\Pr[Z_t = 0 \mid Y_t = 1] \leq \sum_{a=1}^{h_1} \Pr[\xi_{1,a}] \leq \sum_{j=0}^{\infty} (0.4051)^{2^j} \leq 0.597$$

which implies

$$\Pr[Z_t = 1 \mid Y_t = 1] \geq 0.403$$

Consequently

$$\Pr[\text{Task } t \text{ is successfully scheduled}] \geq \frac{\alpha x_t}{2} \cdot 0.403 \cdot \frac{1}{2} = \frac{0.403\alpha}{4} x_t$$

Finally, we obtain

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{0.403\alpha}{4} OPT \geq \frac{1}{310.18} OPT$$

This completes the proof of Theorem 4.7. ∎

### 4.2.2 An Improved Constant for sUfpTree

The sUfpTree is equivalent to the sUfpTree-2Spider. Like the sUfpPath, we can obtain a constant-factor approximation for $\delta$-small instances of the sUfpTree using Theorem 4.4. However, the loose upper bounds used in Theorem 4.4 give us an approximation factor of $1/(822.37 \cdot 2^{2.15}) = 1/3649.91$. We will state a theorem here which attains a better approximation factor. As in Theorem 4.7, the proof is nearly identical to Theorem 4.4 and so only the parts of the calculations which are different are stated here so as to avoid repetition.

**Theorem 4.8** *Consider $\delta = 0.0005$. For $\delta$-small instances of the sUfpTree under the Nba there exists a non-adaptive scheduling algorithm $\mathcal{A}$ for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{1077.59} \cdot \phi(\mathbf{c}+\mathbf{1}) \geq \frac{1}{1077.59} \cdot OPT$$

**Proof.** Along the lines of Theorem 4.7 we will only state here a few calculations which differ from the proof of Theorem 4.4.

As before we define the constants $\alpha = 0.016$ and $\beta = (1/2 - 2\delta - \alpha)/\alpha$. We obtain

$$\Pr[\xi_{i,a}] \leq \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{\alpha c_{e_{i,a}}} \leq (0.2913)^{c_{e_{i,a}}}$$

This leads to

$$\Pr[Z_t = 0 \mid Y_t = 1] \leq \sum_{i=1}^{2}\sum_{a=1}^{h_1} \Pr[\xi_{i,a}] \leq \sum_{i=1}^{2}\sum_{j=0}^{\infty} (0.2913)^{2^j} \leq 0.768$$

which implies

$$\Pr[Z_t = 1 \mid Y_t = 1] \geq 0.232$$

Consequently

$$\Pr[\text{Task } t \text{ is successfully scheduled}] \geq \frac{\alpha x_t}{2} \cdot 0.232 \cdot \frac{1}{2} = \frac{0.232\alpha}{4} x_t$$

Finally, we obtain

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{0.232\alpha}{4} OPT \geq \frac{1}{1077.59} OPT$$

This completes the proof of Theorem 4.8. ∎

# Chapter 5

# Algorithms for Large Tasks

In this section we will give results for $\delta$-large instances of special cases of the sUfpTree-Subtree. These results when combined according to §3 with algorithms in §4 will result in polytime non-adaptive constant-factor approximation algorithms for the various special cases of the sUfpTree-Subtree that we are concerned with. Recall that a task $t$ is $\delta$-large if $\mu_t > \delta b_t$.

## 5.1 Uniform Capacities

When all capacties are equal (and scaled to 1), a task $t$ is $\delta$-large if $\mu_t > \delta$. Lemma 5.1 states that for $\delta$-large instances of the sRapTree-$k$Subtree, $\phi'(\mathbf{1})$ is within a constant factor of $\phi(\mathbf{2})$. Recall that $\phi(\mathbf{u})$ and $\phi'(\mathbf{u})$ are the values of the optimal solutions to the linear programs $\mathcal{L}(\mathbf{u})$ and $\mathcal{L}'(\mathbf{u})$ defined in §2.1.

**Lemma 5.1** *If $\delta \in (0,1)$, then for $\delta$-large instances of the sRapTree-$k$Subtree,*

$$\phi'(\mathbf{1}) \geq \frac{\delta}{2}\phi(\mathbf{2})$$

**Proof.** Let $(x_1, \ldots, x_n)$ be an optimal solution to $\mathcal{L}(\mathbf{2})$. Then note that $(x_1/2, \ldots, x_n/2)$ is a feasible solution to $\mathcal{L}(\mathbf{1})$ under which the value of the objective function is $\phi(\mathbf{2})/2$. Hence

$$\phi(\mathbf{1}) \geq \frac{1}{2}\phi(\mathbf{2})$$

Now let $(x_1, \ldots, x_n)$ be an optimal solution to $\mathcal{L}(\mathbf{1})$. For each task $t$ define $x'_t = \delta x_t$. For any edge $e$ we have that $\sum_{t \in T_e} x'_t = \sum_{t \in T_e} \delta x_t < \sum_{t \in T_e} \mu_t x_t \leq 1$. We infer that the solution $(x'_1, \ldots, x'_n)$ is a feasible solution to $\mathcal{L}'(\mathbf{1})$. Under this solution the objective function has the value $\delta\phi(\mathbf{1})$. Hence

$$\phi'(\mathbf{1}) \geq \delta\phi(\mathbf{1})$$

This completes the proof of Lemma 5.1. ∎

Theorem 5.2 obtains a constant-factor approximation for $\delta$-large instances of the sRapPath and will be used to devise a constant-factor approximation algorithm for the sRapPath.

**Theorem 5.2** *If $\delta \in (0,1)$, then for $\delta$-large instances of the sRapPath, there exists a polytime non-adaptive scheduling algorithm $\mathcal{A}(\delta)$ for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A}(\delta))] \geq \phi'(\mathbf{1}) \geq \frac{\delta}{2} \cdot OPT$$

**Proof.** Consider the constraint matrix corresponding to the linear program $\mathcal{L}'(\mathbf{1})$ defined in §2.1. All entries of this matrix are either 0 or 1. Furthermore, each column $t$ has a contiguous sequence of entries all of which are 1s and the rest of its entries are 0s. The consecutive-ones property implies the existence of an optimal solution to $\mathcal{L}'(\mathbf{1})$ that is also integral; this is because the constraint matrix is totally-unimodular [Sch03]. Let $(x'_1, \ldots, x'_n)$ be such a solution and let $A' = \{t \in T \mid x'_t = 1\}$. Observe that since all the $x'_t$ values are integral, the constraints of $\mathcal{L}'(\mathbf{1})$ imply that the set $A'$ is an independent set of the interval graph underlying the sRapPath instance.

This leads to the following algorithm which we will denote by $\mathcal{A}$: Find a maximum weighted independent set $A$ of the interval graph formed by the set of intervals $P_t = [a_t, b_t)$ having weight $\widetilde{v}_t$ for each task $t$. We can find this in polynomial-time, e.g., using the total-unimodularity result above. Now, schedule the tasks in $A$ in arbitrary order.

Since the intervals corresponding to the tasks in $A$ are disjoint, each edge in $P_t$ will have residual capacity 1 at the time that task $t$ is scheduled. Thus the probability that task $t$ is successfully scheduled is $\Pr[S_t \leq 1]$ and the expected payoff due to task $t$ is $v_t \Pr[S_t \leq 1] = \widetilde{v}_t$. Hence, using Lemma 5.1 we infer that

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] = \sum_{t \in A} \widetilde{v}_t \geq \sum_{t \in A'} \widetilde{v}_t = \sum_{t \in T} \widetilde{v}_t x'_t = \phi'(\mathbf{1}) \geq \frac{\delta}{2}\phi(\mathbf{2}) \geq \frac{\delta}{2} OPT$$

This completes the proof of Theorem 5.2. ∎

Let us now consider the sRapTree-$k$Spider. We will prove in Theorem 5.6 that the integrality gap of $\mathcal{L}'(\mathbf{u})$ for UfpTree-$k$Spider instances with unit demands and integral capacities is $2k$ under the Nba. Using this and Lemma 5.1, we now prove Theorem 5.3 which gives a constant factor approximation for $\delta$-large instances of the sRapTree-$k$Spider under the Nba.

**Theorem 5.3** *If $\delta \in (0, 1)$, then for $\delta$-large instances of the sRapTree-$k$Spider, there exists a polytime non-adaptive scheduling algorithm $\mathcal{A}$ for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{2k} \cdot \phi'(\mathbf{1}) \geq \frac{\delta}{4k} \cdot OPT$$

**Proof.** We will show in Theorem 5.6 that for the sRapTree-$k$Spider, $\mathcal{L}'(\mathbf{1})$ has an integral solution for which the value of the objective function is at least $\phi'(\mathbf{1})/(2k)$. Let $(x'_1, \ldots, x'_n)$ be such a solution and let $A = \{t \in T \mid x'_t = 1\}$. Observe again, as in the proof of Theorem 5.2, that since all the $x'_t$ values are integral, the constraints of $\mathcal{L}'(\mathbf{1})$ imply that the set $A$ is an independent set of the conflict graph underlying the sRapTree-$k$Spider instance.

Observe again, as in Theorem 5.6, that since the tasks in $A$ are edge-disjoint the expected payoff due to task $t$ is $v_t \Pr[S_t \leq 1] = \widetilde{v}_t$. From Lemma 5.1, we infer that

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] = \sum_{t \in A} \widetilde{v}_t = \sum_{t \in T} \widetilde{v}_t x'_t = \frac{1}{2k}\phi'(\mathbf{1}) \geq \frac{\delta}{4k}\phi(\mathbf{2}) \geq \frac{\delta}{4k} OPT$$

This completes the proof of Theorem 5.3. ∎

## 5.2 Non-Uniform Capacities

Lemma 5.4, which is similar to Lemma 5.1, states that for $\delta$-large instances of the sUfpTree-Subtree under the Nba, $\phi'(\lfloor \mathbf{c} \rfloor)$ is within a constant factor of $\phi(\mathbf{c} + \mathbf{1})$. Recall that $\phi(\mathbf{u})$ and $\phi'(\mathbf{u})$ are the values of the optimal solutions to the linear programs $\mathcal{L}(\mathbf{u})$ and $\mathcal{L}'(\mathbf{u})$ defined in §2.1. We will use Lemma **??** to obtain a constant-factor approximation for $\delta$-large instances of the sUfpPath and the sUfpTree-$k$Spider under the Nba.

**Lemma 5.4** *If $\delta \in (0, 1)$. then for $\delta$-small instances of the sUfpTree-Subtree under the Nba,*

$$\phi'(\lfloor \mathbf{c} \rfloor) \geq \frac{\delta}{3} \cdot \phi(\mathbf{c} + \mathbf{1})$$

**Proof.** Let $(x_1, \ldots, x_n)$ be an optimal solution to $\mathcal{L}(\mathbf{c}+\mathbf{1})$. Then note that $(x_1/3, \ldots, x_n/3)$ is a feasible solution to $\mathcal{L}(\lfloor \mathbf{c} \rfloor)$ since for all $e \in E$, $c_e \geq 1$ and consequently $\lfloor c_e \rfloor \geq (c_e+1)/3$. The value of the objective function under this solution is $\sum_{t \in T} \widetilde{v}_t x_t/3 = \phi(\mathbf{c}+\mathbf{1})/3$. Hence

$$\phi(\lfloor \mathbf{c} \rfloor) \geq \frac{1}{3}\phi(\mathbf{c}+\mathbf{1})$$

Note that $\mu_t > \delta b_t \geq \delta$. It follows from an argument similar to the one used in Lemma 5.1 that

$$\phi'(\lfloor \mathbf{c} \rfloor) \geq \delta \phi(\lfloor \mathbf{c} \rfloor)$$

This completes the proof of Lemma 5.4. ∎

Theorem 5.5 gives an approximation for $\delta$-large instances of the sUfpPath under the Nba. Notably, it guarantees that the payoff (and not its expectation) is within at most a constant-factor of the optimal payoff.

**Theorem 5.5** *If $\delta \in (0,1)$, then for $\delta$-large instances of the sUfpPath under the Nba there exists a non-adaptive scheduling algorithm $\mathcal{A}$ for which*

$$\mathcal{P}(\mathcal{A}) \geq \phi'(\lfloor \mathbf{c} \rfloor) \geq \frac{\delta}{3} \cdot OPT$$

**Proof.** Consider the constraint matrix corresponding to the linear program $\mathcal{L}'(\lfloor \mathbf{c} \rfloor)$ defined in §2.1. As in Theorem 5.2 we infer from the consecutive-ones property that this matrix is totally-unimodular and consequently that it has an optimal solution which is integral [Sch03]. Let $(x'_1, \ldots, x'_n)$ be such a solution and let $A = \{t \in T \mid x'_t = 1\}$.

Consider the algorithm $\mathcal{A}$ which schedules the tasks in $A$ in arbitrary order. Since we are operating under the Nba and since we scaled the capacities so that $c_{\min} = 1$, we know that $S_t \leq 1$ . Consequently, we have that for every edge $e$

$$\sum_{t \in A_e} S_t = \sum_{t \in T_e} S_t x'_t \leq \sum_{t \in T_e} x'_t \leq \lfloor c_e \rfloor$$

Hence, it is guaranteed that every task $t$ scheduled by $\mathcal{A}$ will be successfully scheduled and will generate payoff $v_t$. From Lemma 5.4, we infer that

$$P(\mathcal{A}) = \sum_{t \in A} v_t = \sum_{t \in T} v_t x'_t = \phi'(\lfloor \mathbf{c} \rfloor) \geq \frac{\delta}{3}\phi(\mathbf{c}+\mathbf{1}) \geq \frac{\delta}{3} OPT$$

This completes the proof of Theorem 5.5. ∎

We will now proceed to show an appromixation to $\delta$-large instances of the sUfpTree-$k$Spider. Towards this end, Theorem 5.6 establishes a constant-factor approximation algorithm under the Nba for the UfpTree-$k$Spider with unit demands and integral capacities. This algorithm is a straightforward generalization of the approximation algorithm for the UfpTree with unit-demands and integral capacities in the Multicommodity Flow paper by Chekuri, Mydlarz and Shephard [CMS07]. We note in passing that Theorem 5.6, along with Theorem 3.1 in the aforementioned paper [CMS07] implies a constant-factor approximation for the UfpTree-$k$Spider with arbitrary integral capacities and integral demands under the Nba.

**Theorem 5.6** *For the sUfpTree-$k$Spider, then the integrality gap of $\mathcal{L}'(\mathbf{c})$ is at most $2k$. Furthermore, there exists a polytime algorithm which can find a $2k$-approximate integral solution to $\mathcal{L}'(\mathbf{c})$.*

**Proof.** Let $\mathbf{x}' = (x'_1, \ldots, x'_n)$ be an optimal solution to $\mathcal{L}'(\mathbf{c})$. Let $h$ be an integer such that $h\mathbf{x}'$ is an integral vector. [1] Consider a set of tasks $\widetilde{T}$ which has $hx'_t$ replicas ef every task $t$ in $T$. We will show that there exists a $2kh$-colouring of tasks in $\widetilde{T}$ each of which is feasible such that no two edge replicas are coloured the

[1] For any fixed $\epsilon > 0$, one could alter $\mathbf{x}'$ such that (i)$k\mathbf{x}'$ is integral, (ii) $k$ is polynomially bounded and (iii)$\mathbf{w} \cdot \mathbf{x}' \geq (1-\epsilon)\mathbf{w} \cdot \mathbf{x}$

same. Assuming that we have shown this, we infer that the sum of the payoffs of these $2kh$ colourings equals $h\phi'(\mathbf{c})$. Hence, the feasible set of tasks corresponding to at least one of these colourings must have payoff $\phi'(\mathbf{c})/(2k)$. It follows that $\mathcal{L}'(\mathbf{c})$ has an integrality gap of at most $2k$.

To show the existence of such a $2kh$-colouring, we claim the following:

**Claim 5.7** *Consider a capacitated graph $G' = (V', E')$ which is a rooted tree and a set of tasks $T'$ such that each task $t \in T'$ corresponds to the $k$-*spider *$P'_t$ in $G'$. If*

*(i)* $| T'_e | \leq hc_e$ *for all* $e \in E'$.

*(ii) For each leaf $l$ having edge $e$ connecting it to its parent, the set of tasks $T'_e$ are partitioned into at most $c_e$ bins at $l$, each having size between $[1, 2h)$.*

*(iii) The leaves of the $k$-*spiders *corresponding to each task are all distinct leaves of $G'$ and the root has exactly one child.*

*then there exists a colouring of $T'$ which*

*(iv) Uses at most $2kh$ colours.*

*(v) Ensures that for each leaf, all the tasks in each bin are all coloured differently.*

*(vi) Ensures that for each colour $d$ and each edge $e$, $| \{t \in T'_e \mid colour(t) = d\} | \leq c_e$.*

Assuming that Claim 5.7 is true, we now apply it to the set of tasks $\widetilde{T}$ on the graph $G$. Observe that $\widetilde{T}$ satisfies (i) since $\mathbf{x}'$ satisfies the capacity constraints (L'.1) of the linear program $\mathcal{L}'(\mathbf{c})$ defined in §2.1. We know that $\widetilde{T}$ satisfies (iii) from the assumptions we made in §1.2 without loss of generality. We now need to partition the tasks at each leaf into bins such that (ii) is satisfied and the colouring guaranteed by Claim 5.7 does not have any two edge replicas colured the same. To ensure the latter, we partition the tasks into bins such that all replicas of each task $t \in T$ fall into the same bin at all endpoints of their corresponding $k$-*spiders*. Consider groups of replicas of tasks incident to leaf $l$, connected to its parent with edge $e$. We consider these groups sequentially in arbitrary order and group them into bins at $l$, starting a new bin when the size of the current bin equals or exceeds $h$. We know from (i) that the $| \widetilde{T}_e | \leq hc_e$. Since the size of each group of replicas is at most $h$, the binning procedure described above ensures that all bins except possibly the last one have sizes in the range $[h, 2h)$ and hence that the number of bins at leaf $l$ is at most $c_e$. We conclude that this binning process satisfies (ii). Furthermore, we infer from (v) that the colouring guaranteed by Claim 5.7 ensures that all replicas of every task are coloured differently, completing the proof of Theorem 5.6. ∎

Let us now prove Claim 5.7.

**Claim 5.7** *Consider a capacitated graph $G' = (V', E')$ which is a rooted tree and a set of tasks $T'$ such that each task $t \in T'$ corresponds to the $k$-*spider *$P'_t$ in $G'$. If*

(i) $| T'_e | \leq hc_e$ *for all* $e \in E'$.

(ii) *For each leaf $l$ having edge $e$ connecting it to its parent, the set of tasks $T'_e$ are partitioned into at most $c_e$ bins at $l$, each having size between $[1, 2h)$.*

(iii) *The leaves of the $k$-*spiders *corresponding to each task are all distinct leaves of $G'$ and the root has exactly one child.*

*then there exists a colouring of $T'$ which*

(iv) *Uses at most $2kh$ colours.*

(v) *Ensures that for each leaf, all the tasks in each bin are all coloured differently.*

(vi) *Ensures that for each colour $d$ and each edge $e$,* $\mid \{t \in T'_e \mid colour(t) = d\} \mid \leq c_e$.

**Proof.** We induct on the number of vertices in the tree. Our base case occurs when the root is the only non-leaf node in the tree. In this case, since it has exactly one child, there must be a single leaf $l$ in the tree. This implies that all the *k-spiders* must be 1-*spiders*. Let $e$ denote the only edge in the tree. Let $B_1, \ldots, B_r$ denote the bins at $l$. Colour the tasks in bin $B_i$ with colours $1, \ldots, \mid B_i \mid$. Clearly, this colouring satisfies (v). Since the sizes of the bins are less than $2h$, we infer that it also satisfies (iv). Finally, since $r$ is less than or equal to $c_e$, we infer that it satisfies (vi).

Otherwise, consider $v$, a deepest non-leaf node in the tree. Let us denote its set of its children, all of which are leaves, by $L = \{l_1, \ldots, l_z\}$. We collapse all the children of $v$ and $v$ itself into a single vertex to obtain a smaller instance $G'' = (V'', E'')$ and a set of tasks $T''$. Here $V'' = V' \setminus L$. $E''$ is the restriction of $E'$ to $V''$. Let $T'_L$ denote set of those tasks $t \in T'$ for which all leaves of $P'_t$ belong to $L$. Then $T'' = T' \setminus T'_L$. The vertex $v$ is a leaf in $G''$. Let $e$ denote the edge connecting $v$ to its parent. To partition $T''_e$ into the bins at $v$ in the smaller instance, we consider the set of sets $B_e \setminus T'_L$ for each bin $B$ at each leaf in $L$. Let us denote this set of sets by $U$. First, we observe that the sets in $U$ are mutually disjoint. This is because every task $t$ in $T''_e$ must be such that $P'_t$ has exactly one of its leaves in $L$, since if $P'_t$ had more than one leaf belonging to $L$, then the definition of a *k-spider* would imply that all of its leaves would necessarily belong to $L$. Having made this observation, we create a new bin at $v$ for all sets in $U$ whose sizes are in the range $[h, 2h)$. We will combine the sets in $U$ having size $[1, h)$ into bins at $v$. We combine these serially in arbitrary order and start a new bin at $v$ once the current bin's size equals or exceeds $h$. We stop when we run out of bins in $U$ to combine. This process guarantees that each bin at $v$, except possibly the last one, has size in the range $[h, 2h)$. We know from (i) that $\mid T''_e \mid = \mid T'_e \mid \leq hc_e$. Hence the number of bins at $v$ is at most $c_e$ and we infer that the smaller instance satisfies (ii). It trivially satisfies (i) since $T'' \subseteq T'$. The smaller instance satisfies (iii) since we replaced exactly one leaf of each task $t \in T'_e$ with $v$ which was not a leaf in $G'$.

From the inductive hypothesis we know that there exists a $2kh$-colouring for the set of tasks $T''$ in the smaller instance $G''$ which satisfies (iv), (v) and (vi). We extend this colouring to $T'$, letting the colours assigned to tasks in $T''$ remain unchanged and appropriately assigning colours to the tasks in $T'_L$. We know that for every bin $B$ at every leaf in $L$ the tasks in $B_e \setminus T'_L$ were all assigned to the same bin at $v$, and hence were coloured differently. We conclude that all tasks in $T''$ which fall in the same bin at some leaf in $L$ are coloured differently. It remains to colour the tasks in $T'_L$ so that (iv) and (v) are satisfied. Consider tasks in $T'_L$ in any order. The *k-spider* $P_t$ for each such task $t$ has at most $k$ leaves. Task $t$ falls into some bin at each of these leaves. Each such bin has at most $2h - 1$ tasks and since $t$ is not yet coloured, the total number of colours used by all the bins together is at most $k(2h - 2) = 2kh - 2k$. We colour $t$ with any one of the $2k$ unused colours, thereby ensuring that this colouring of tasks in $T'$ satisfies (iv) and (v). It follows from the inductive hypothesis that this colouring of tasks in $T'$ satisfies (vi) for all edges except those connecting a vertex in $L$ to $v$. Let $e_i$ denote the edge between $v$ and $l_i$. Each task in $T'_{l_i}$ must be in some bin at $l_i$. Since there are at most $c_{e_i}$ bins at $l_i$ and the tasks in each bin are coloured differently, we infer that there are at most $c_{e_i}$ tasks of any given colour in $T'_{l_i}$. Thus, this coloring satisfies (vi). This completes the induction and the proof of Claim 5.7.

We now use Theorem 5.6 to obtain a constant-factor approximation algorithm for $\delta$-large instances of the sUfpTree-$k$Spider. Note, once again, that Theorem 5.8 guarantees that the payoff (and not its expectation) is within at most a constant-factor of the optimal payoff.

**Theorem 5.8** *If $\delta \in (0, 1)$, then for the sUfpTree-kSpider under the Nba, there exists a polytime non-adaptive scheduling algorithm $\mathcal{A}$ for which*

$$\mathcal{P}(\mathcal{A}) \geq \frac{1}{2k} \cdot \phi'(\lfloor \mathbf{c} \rfloor) \geq \frac{\delta}{6k} \cdot OPT$$

**Proof.** We have shown in Theorem 5.6 that $\mathcal{L}'(\lfloor \mathbf{c} \rfloor)$ has an integral solution for which the value of the objective function is at least $\phi'(\lfloor \mathbf{c} \rfloor)/(2k)$. Let $(x'_1, \ldots, x'_n)$ be such a solution and let $A = \{t \in T \mid x'_t = 1\}$.

Consider the algorithm $\mathcal{A}$ which schedules the tasks in $A$ in arbitrary order. Since we are operating under the Nba and since we scaled the capacities so that $c_{\min} = 1$, we know that $S_t \leq 1$. Consequently, we have

that for every edge $e$

$$\sum_{t \in A_e} S_t = \sum_{t \in T_e} S_t x'_t \leq \sum_{t \in T_e} x'_t \leq c_e$$

Hence, it is guaranteed that every task $t$ scheduled by $\mathcal{A}$ will be successfully scheduled and will generate payoff $v_t$. From Lemma 5.4

$$P(\mathcal{A}) = \sum_{t \in A} v_t = \sum_{t \in T} v_t x'_t \geq \frac{1}{2k} \phi'(\lfloor \mathbf{c} \rfloor) \geq \frac{\delta}{6k} \phi(\mathbf{c} + \mathbf{1}) \geq \frac{\delta}{6k} OPT$$

This completes the proof of Theorem 5.8. □

## 5.3 sRap$k$Tree-Subtree: An approximation to non-adaptive algorithms

This section gives a polytime constant-factor approximation to non-adaptive algorithms for the sRap$k$Tree-Subtree. It is separated from §5.1 even though it deals with uniform capacities because the argument is of a different nature, providing an approximation not to a linear program but directly to an optimal non-adaptive algorithm.

Let $OPT_N$ denote the expected payoff of an optimal non-adaptive algorithm. We will first sketch a $O(mn2^k)$ algorithm $\mathcal{A}$ which finds an optimal integral solution to $\phi'(\mathbf{1})$ and then show that the expected payoff of the algorithm which schedules the tasks selected by $\mathcal{A}$ in arbitrary order is within a constant factor of $OPT_N$.

**Theorem 5.9** *There exists an algorithm for the sRap$k$Tree-Subtree which finds an optimal integral solution to $\phi'(\mathbf{1})$ and runs in time $O((m+n)2^k + mn)$.*

**Proof.** Note that an optimal integral solution to $\phi'(\mathbf{1})$ is also an optimal solution to the sRap$k$Tree-Subtree instance on the same capacitated graph in which each task $t$ has deterministic payoff $\widetilde{v}_t$. We will denote the tasks in this deterministic instance by $\mathcal{T}$. Further, for $v \in V$ we denote by $\mathcal{T}_v$ the set of tasks $t$ in $\mathcal{T}$ such that $P_t$ is completely contained in the subtree rooted at $v$.

Let $dp(v,t)$ denote the payoff of the optimal solution for the set of tasks $\mathcal{T}_v$ where the edge connecting $v$ to its parent is covered with task $t \in \mathcal{T} \cup \{0\}$. Here $dp(v,0)$ denotes the optimal payoff when the edge connecting $u$ to its parent is unused. The payoff of task $t$ is not included in the $dp$ value.

Denote the set of children of vertex $v$ by $C_v$ and the tasks in $\mathcal{T}_v \setminus \bigcup_{c \in C_v} \mathcal{T}_c$ by $t_{v,1} \ldots t_{v,n_v}$. Note that $\{t_{v,1} \ldots t_{v,n_v}\}$ is the set of tasks in $\mathcal{T}$ having $v$ as root. For $v \in V$, $i \in [0, n_v]$, $C \subseteq C_v$, let $cdp(v,i,C)$ denote the payoff of the optimal solution for those among the first $i$ tasks having $v$ as root which are completely contained (besides vertex $v$) in the subtrees rooted at vertices in $C$ and those tasks in $\mathcal{T}$ which are completely contained in the subtrees rooted at vertices in $C$. Formally $cdp(v,i,C)$ is the optimal solution for the set of tasks $\left(\bigcup_{c \in C} \mathcal{T}_c\right) \cup \{t_{v,i'} \mid i' \in [i] \text{ and } C_v \cap P_{t_{v,i'}} \subseteq C\}$.

We have the recurrences:

$$cdp(v,0,C) = \sum_{c \in C} dp(c,0)$$

$$cdp(v,i,C) = \begin{cases} cdp(v,i-1,C) & \text{if } C_v \cap P_{t_{v,i}} \not\subseteq C \\ \max\Big(cdp(v,i-1,C), & \\ \quad cdp(v,i-1,C \setminus P_{t_{v,i}}) + w_{t_{v,i}} + \sum_{c \in C_v \cap P_{t_{v,i}}} dp(c, t_{v,i})\Big) & \text{if } C_v \cap P_{t_{v,i}} \subseteq C \end{cases}$$

$$dp(v,t) = cdp(v, n_v, C_v \setminus P_t) + \sum_{c \in C_v \cap P_t} dp(c,t)$$

The recurrences above results in a $O((m+n)2^k + mn)$ using precomputation of the summations in the computation of the $cdp$ values. The payoff of the optimal solution is $dp(r,0)$ where $r$ is the root. We can find an optimal solution by tracing the $dp$ and $cdp$ values.

**Lemma 5.10** *Consider a rooted $k$-ary tree $G' = (V', E')$ and a set of subtrees $T'$ of $G'$. If $i \in N$ is such that $\mid T'_e \mid \leq i$ for all $e \in E$ then $T'$ can be partitioned into at most $k(i-1)+1$ edge-disjoint subsets.*

**Proof.** The proof is constructive. Without loss of generality, assume that the tasks in $T'$ are numbered $1, \ldots, n$ in non-decreasing order of the depth in $G$ of their least-depth vertex. In the $j^{\text{th}}$ iteration, initialize the current partition $Q_j$ as the empty set. Iterate over the unpartitioned tasks in increasing order of their indices and add each such task to partition $Q_j$ if adding it does not violate the edge-disjointness of $Q_j$. Stop when all tasks have been partitioned.

If there are more than $k(i-1)+1$ iterations, this means that at least one task $t$ must be unclassified after the completion of $k(i-1)+1$ iterations. Task $t$ was not partitioned in the $j^{\text{th}}$ iteration because some edge in $P_t$ was already used by $Q_j$. Let $e_j$ be one such edge. If there are multiple candidates, choose any one with the least depth. Let $v$ be the least-depth vertex of $P_t$. Since each iteration considers the tasks in increasing order, $e_j$ must be one of the edges from the $v$ to its children. Since $G$ is a $k$-ary tree there are at most $k$ such edges. The box principle implies that at least $i$ of the elements in the sequence $(e_j)_{j \in [k(i-1)+1]}$ values must be identical. Since the $Q_j$'s are mutually disjoint, we infer that $i$ tasks besides task $t$ all pass through some edge $e$ in $P_t$. Thus $\mid T'_e \mid \geq i+1$, a contradiction. This completes the proof of Lemma 5.10. ∎

**Theorem 5.11** *Consider $\delta = 0.6$. For $\delta$-large instances of the sRapkTree-Subtree there exists a polytime non-adaptive scheduling algorithm $\mathcal{A}$ for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{225k+26} \cdot OPT_N$$

**Proof.** Consider a non-adaptive algorithm $\mathcal{A}'$. Let $t_1, \ldots, t_s$ be the sequence of tasks that it schedules, skipping only past the ones which become infeasible in the process. Let us the denote by $T^i$ the set of the first $i$ tasks scheduled by $\mathcal{A}'$. Formally $T^i = \{t_{i'} \mid i' \in [i]\}$. Consider that the tasks scheduled stack up over each other at each edge, independently of the other edges. We define the level $l_i$ of task $t_i$ as the number of tasks stacked up at the most stacked up edge in $P_{t_i}$ just after scheduling the task $t_i$. Formally $l_i = \max\{\mid T^i_e \mid\mid e \in P_{t_i}\}$. Let $L^i$ denote the set of tasks at level $i$. Let us define the random variable $\mathcal{P}_i(\mathcal{A}')$ as the payoff from tasks in $L^i$ under the scheduling scheme of $\mathcal{A}'$.

From the linearity of expectation, we have that

$$\mathbf{E}[\mathcal{P}(\mathcal{A}')] = \sum_{i \geq 1} \mathbf{E}[\mathcal{P}_i(\mathcal{A}')]$$

Note that $\mid L^i_e \mid \leq i$ for all $e$, since otherwise the last of the tasks in $L^i_e$ to be scheduled would be at level $i+1$ or more, a contradiction. From Lemma 5.10 it follows that the set of tasks in $L_i$ can be partitioned into at most $k(i-1)+1$ (denoted hereafter by $f(i)$) edge-disjoint sets. We know that for all tasks $t$, $\mu_t = \mathbf{E}[\widetilde{S}_t] \geq 0.6$. We know that $\widetilde{S}_t \leq 1$. If $\Pr[\widetilde{S}_t \leq 1/2] > 0.8$, then we would have $\mu_t < 0.8 \cdot \frac{1}{2} + 0.2 \cdot 1 = 0.6$, a contradiction. Hence we know that $\Pr[\widetilde{S}_t \leq 1/2] \leq 0.8$ for all tasks $t$. Let us define the constant $c = 0.8$.

Tasks in $L^1$ are successfully scheduled if and only if their size is at most 1. Hence

$$\mathbf{E}[\mathcal{P}_1(\mathcal{A}')] = \sum_{t \in L^1} v_t \Pr[S_t \leq 1] = \sum_{t \in L^1} \widetilde{v}_t$$

Let $F$ denote the value of the objective for the optimal integral solution to $\phi'(\mathbf{1})$. The set of tasks in $L^1$ is edge-disjoint since $f(1) = 1$ and hence

$$\mathbf{E}[\mathcal{P}_1(\mathcal{A}')] \leq F$$

For $i \geq 2$, a task $t$ in $L^i$ is successfully scheduled only if its size is less than or equal to 1 and at least $i-2$ of the $i-1$ tasks stacked below it on the most stacked-up edge (let us call it $e$) in $P_t$ at the time of its scheduling are of size at most $1/2$. This is because if two of these $i-1$ tasks have size more than $1/2$ then the remnant capacity of $e$ will be 0 and task $t$ will not be successfully scheduled. These two events are independent. If we use the union bound to upper bound the probability of the second event, we infer that $\Pr[\text{Task } t \text{ is successfully scheduled}] \leq \Pr[S_t \leq 1] \cdot \binom{i-1}{i-2} c^{i-2}$. Note that we used the fact that $\Pr[\widetilde{S}_t \leq 1/2] = \Pr[S_t \leq 1/2] \leq c$ for all tasks $t \in T$, which was derived above. Hence

$$\mathbf{E}[\mathcal{P}_i(\mathcal{A}')] \leq \sum_{t \in L^i} v_t \Pr[S_t \leq 1] \cdot \binom{i-1}{i-2} c^{i-2} = (i-1)c^{i-2} \cdot \sum_{t \in L^i} \widetilde{v}_t$$

We know that the tasks in $L^i$ can be partitioned into at most $f(i)$ edge-disjoint subsets. Hence

$$\mathbf{E}[\mathcal{P}_i(\mathcal{A}')] \leq (i-1)c^{i-2} \cdot f(i) \cdot F$$

We use these bounds to bound the payoff of $\mathcal{A}'$.

$$\begin{aligned}
\mathbf{E}[\mathcal{P}(\mathcal{A}')] &= \sum_{i \geq 1} \mathbf{E}[\mathcal{P}_i(\mathcal{A}')] \\
&\leq F + \sum_{i \geq 2} \left( (i-1)c^{i-2} \cdot f(i) \cdot F \right) \\
&= F + \sum_{i \geq 2} \left( (i-1)c^{i-2} \cdot (k(i-1)+1) \cdot F \right) \\
&= F + F \cdot \sum_{i \geq 0} \left( (i+1)c^i \right) + F \cdot k \sum_{i \geq 0} \left( (i+1)^2 c^i \right) \\
&= F \cdot \left( 1 + \frac{1}{(1-c)^2} + k\left( \frac{1}{(1-c)^2} + \frac{2c}{(1-c)^3} \right) \right) \\
&= (225k + 26)F
\end{aligned}$$

We have already shown in Theorem 5.9 that we can find an optimal integral solution $\mathbf{x}'$ to $\phi'(\mathbf{1})$ in time $O((m+n)2^k + mn)$. Consider the set $A = \{t \in T \mid x'_i = 1\}$. The algorithm $\mathcal{A}$ schedules tasks in $A$ in arbitrary order. The constrants of $\mathcal{L}'(\mathbf{1})$ are such that the tasks in $A$ are edge-disjoint. Hence

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] = \sum_{t \in A} v_t \Pr[s_t \leq 1] = \sum_{t \in A} \widetilde{v}_t = \sum_{t \in T} \widetilde{v}_t x'_t = F$$

We infer that for every non-adaptive algorithm $\mathcal{A}'$,

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{225k + 26} \cdot \mathbf{E}[\mathcal{P}(\mathcal{A}')]$$

Hence

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{225k + 26} \cdot OPT_N$$

This completes the proof of Theorem 5.11.

# Chapter 6

# Approximation Algorithms

In this section, we combine the algorithms in §4 and §5 according to the strategy described in §3 to obtain non-adaptive polytime approximation algorithms for the various special cases of sUfpTree-Subtree that we are interested in.

**Theorem 6.1** *There exists a non-adaptive polytime algorithm $\mathcal{A}$ for the sRapPath for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{36} \cdot OPT$$

*This implies that the adaptivity gap of the sRapPath is at most* 36.

**Proof.** We know from Theorem 4.1, by choosing $q = 1/2$, that for $\delta \in (0, 1/2)$ there exists an algorithm $\mathcal{A}_S(\delta)$ which for $\delta$-small instances of the sRapPath guarantees an expected payoff of at least $\phi(\mathbf{2}) \cdot (1-2\delta)/16 \geq OPT \cdot (1-2\delta)/16$. Theorem 5.2 implies that for $\delta$-large instances of sRapPath, there exists an algorithm $A_L(\delta)$ which guarantees an expected payoff of at least $\phi(\mathbf{2}) \cdot \delta/2 \geq OPT \cdot \delta/2$. We infer from Theorem 3.1 that there exists an algorithm $\mathcal{A}$ for sRapPath which guarantees an expected payoff of at least $OPT \cdot 1/(16/(1-2\delta)+2/\delta)$. This quantity attains its maximum value in the interval $(0, 1/2)$ at $\delta = 1/6$ which equals $OPT \cdot 1/36$. ∎

**Theorem 6.2** *There exists a non-adaptive polytime algorithm $\mathcal{A}$ for the sUfpPath under the Nba for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{6310.18} \cdot OPT$$

*This implies that the adaptivity gap of the sUfpPath under the Nba is at most* 6310.18.

**Proof.** Consider $\delta = 0.0005$. We know from Theorem 4.7, that there exists an algorithm $\mathcal{A}_S$ which for $\delta$-small instances of the sUfpPath under the Nba guarantees an expected payoff of at least $\phi(\mathbf{c}+\mathbf{1}) \cdot 1/310.18 \geq OPT \cdot 1/310.18$. Theorem 5.5 implies that for $\delta$-large instances of the sUfpPath under the Nba, there exists an algorithm $A_L$ which guarantees a payoff of at least $\phi'(\mathbf{1}) \cdot \delta/3 \geq OPT \cdot \delta/3$. We infer from Theorem 3.1 that there exists an algorithm $\mathcal{A}$ for the sUfpPath under the Nba which guarantees an expected payoff of at least $OPT \cdot 1/(310.18 + 3/\delta) = OPT \cdot 1/6310.18$. ∎

**Theorem 6.3** *There exists a non-adaptive polytime algorithm $\mathcal{A}$ for the sUfpTree under the Nba for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{25077.59} \cdot OPT$$

*This implies that the adaptivity gap of the sUfpTree under the Nba is at most* 25077.59.

**Proof.** Consider $\delta = 0.0005$. We know from Theorem 4.8, that there exists an algorithm $\mathcal{A}_S$ which for $\delta$-small instances of the sUfpTree under the Nba guarantees an expected payoff of at least $\phi(\mathbf{c}+\mathbf{1}) \cdot 1/1077.59 \geq OPT \cdot 1/1077.59$. Theorem 5.8 implies that for $\delta$-large instances of the sUfpTree under the Nba, there exists an algorithm $A_L$ which guarantees a payoff of at least $\phi'(\mathbf{1}) \cdot \delta/12 \geq OPT \cdot \delta/12$. We infer from Theorem 3.1 that there exists an algorithm $\mathcal{A}$ for the sUfpTree under the Nba which guarantees an expected payoff of at least $OPT \cdot 1/(1077.59 + 12/\delta) = OPT \cdot 1/25077.59$. ∎

**Theorem 6.4** *There exists a non-adaptive polytime algorithm $\mathcal{A}$ for the sRapTree-$k$Spider for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{48k} \cdot OPT$$

*This implies that the adaptivity gap of the sRapTree-$k$Spider is at most* $48k$.

**Proof.** We know from Theorem 4.1, by choosing $q = 1/2$, that there exists an algorithm $\mathcal{A}_S(\delta)$ which for $\delta$-small instances of the sRapTree-$k$Spider guarantees an expected payoff of at least $\phi(\mathbf{2}) \cdot (1 - 2\delta)/16k \geq OPT \cdot (1 - 2\delta)/16k$. Theorem 5.3 implies that for $\delta$-large instances of sRapTree-$k$Spider, there exists an algorithm $A_L$ which guarantees an expected payoff of at least $\phi'(\mathbf{1}) \cdot \delta/(4k) \geq OPT \cdot \delta/(4k)$. We infer from Theorem 3.1 that there exists an algorithm $\mathcal{A}$ for the sRapTree-$k$Spider which guarantees an expected payoff of at least $OPT \cdot 1/(4k/\delta + 16k/(1 - 2\delta))$. At $\delta = 1/4$ this quantity equals $OPT \cdot 1/48k$. ∎

**Corollary 6.5** *There exists a non-adaptive polytime algorithm $\mathcal{A}$ for the sRapTree for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{96} \cdot OPT$$

*This implies that the adaptivity gap of the sRapTree is at most* 96.

**Theorem 6.6** *There exists a non-adaptive polytime algorithm $\mathcal{A}$ for the sUfpTree-$k$Spider under the Nba for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{822.37k^{2.15} + 12000k} \cdot OPT$$

*This implies that the adaptivity gap of sUfpTree-$k$Spider under the Nba is at most* $822.37k^{2.15} + 12000k$.

**Proof.** Consider $\delta = 0.0005$. We know from Theorem 4.4 that there exists an algorithm $\mathcal{A}_S$ which for $\delta$-small instances of the sUfpTree-$k$Spider under the Nba guarantees an expected payoff of at least $\phi(\mathbf{c}+\mathbf{1}) \cdot 1/(822.37k^{2.15}) \geq OPT \cdot 1/(822.37k^{2.15})$. Theorem 5.8 implies that for $\delta$-large instances of the sUfpTree-$k$Spider under the Nba, there exists an algorithm $A_L$ which guarantees a payoff of at least $\phi'(\mathbf{c}+\mathbf{1}) \cdot \delta/(6k) \geq OPT \cdot \delta/(6k)$. We infer from Theorem 3.1 that there exists an algorithm $\mathcal{A}$ for the sUfpTree-$k$Spider under the Nba which guarantees an expected payoff of at least $OPT \cdot 1/(6k/\delta + 822.37k^{2.15}) = OPT \cdot 1/(12000k + 822.37k^{2.15})$ ∎

**Theorem 6.7** *There exists a non-adaptive polytime algorithm $\mathcal{A}$ for the* sRap$k$Tree-Subtree *for which*

$$\mathbf{E}[\mathcal{P}(\mathcal{A})] \geq \frac{1}{257k+26} \cdot OPT_N$$

**Proof.** We know from Theorem 4.1, by choosing $p = 1/2$, that there exists an algorithm $\mathcal{A}_S(\delta)$ which for $\delta$-small instances of sRap$k$Tree-Subtree guarantees an expected payoff of at least $\phi(\mathbf{2}) \cdot (1-2\delta)/16k \geq OPT_N \cdot (1-2\delta)/16k$. Theorem 5.11 implies that for $\delta$-large instances of the sRap$k$Tree-Subtree, there exists an algorithm $A_L$ which guarantees an expected payoff of at least $OPT_N \cdot 1/(225k+16)$. We infer from Theorem 3.1 that there exists an algorithm $\mathcal{A}$ for the sRap$k$Tree-Subtree which guarantees an expected payoff of at least $OPT \cdot 1/(225k + 26 + 16k/(1-2\delta))$. At $\delta = 1/4$, this quantity equals $OPT_N \cdot 1/(257k+26)$. ∎

**Theorem 6.8** *There exists a polytime algorithm $\mathcal{A}$ for the* UfpTree-k*Spider with integral demands and integral capacities under the* Nba *for which*

$$\mathcal{P}(\mathcal{A}) \geq \frac{1}{23.10k} \cdot OPT$$

**Proof.** As we remarked in §5, Theorem 5.6 along with Theorem 3.1 in the paper on the MulticommodityFlow problem on Trees by Chekuri, Mydlarz and Shephard [CMS07] guarantees a polytime approximation algorithm $\mathcal{A}$ for which $\mathcal{P}(\mathcal{A}) \geq 1/2k \cdot 1/11.55 \cdot OPT = 1/23.10k \cdot OPT$. ∎

# Chapter 7

# Concluding Remarks

The following table provides a summary of the results obtained in §6. Note that we have referred to all the approximation factors, even those which are polynomial in $k$ as constant factors in the previous sections since we consider $k$ to be a part of the problem specification.

| Problem | Nba? | Order of Approximation | Approximation Factor | Approximated Payoff |
|---|---|---|---|---|
| sRapPath | No | $O(1)$ | 36 | Optimal |
| sRapTree | No | $O(1)$ | 96 | Optimal |
| sRapTree-$k$Spider | No | $O(k)$ | $48k$ | Optimal |
| sRap$k$Tree-Subtree | No | $O(k)$ | $257k + 26$ | Non-adaptive Optimal |
| sUfpPath | Yes | $O(1)$ | 6310.18 | Optimal |
| sUfpTree | Yes | $O(1)$ | 25077.59 | Optimal |
| sUfpTree-$k$Spider | Yes | $O(k^{2.15})$ | $822.37k^{2.15} + 12000k$ | Optimal |
| UfpTree-$k$Spider | Yes | $O(k)$ | $23.10k$ | Optimal |

An open question is whether the polytime approximation to non-adaptive algorithms for the sRap$k$Tree-Subtree obtained by Theorem 6.7 is also an approximation to adaptive algorithms for it. All our algorithms are offline algorithms. A direction of interest is the proving or disproving of the existence of online algorithms with similar guarantees. Improving the large constant factors and obtaining approximation factors independent of $k$ for the sRapTree-$k$Spider, the sRap$k$Tree-Subtree, the sUfpTree-$k$Spider under the Nba and the UfpTree-$k$Spider under the Nba also poses an interesting challenge. It may be possible to extend the argument for the sRap$k$Tree-Subtree to the sRapTree-$k$Subtree by modifying the dynamic programming argument in Theorem 5.9. Obtaining approximations for the the arbitrary capacity variants of the sUfp without the Nba is a significant challenge with potential scope for plenty of research. Obtaining results for the sUfp on general graphs, such as the work of Chawla and Roughgarden for the single source sUfp [CR06], is another challenge with significant research potential.

# Bibliography


[AGLW14] Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing $2+\varepsilon$ approximation for unsplittable flow on a path. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 26–41, 2014.

[BSW11] Paul S. Bonsma, Jens Schulz, and Andreas Wiese. A constant factor approximation algorithm for unsplittable flow on paths. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 47–56, 2011.

[CCGK07] Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47(1):53–78, 2007.

[CCKR11] Gruia Călinescu, Amit Chakrabarti, Howard J. Karloff, and Yuval Rabani. An improved approximation algorithm for resource allocation. *ACM Transactions on Algorithms*, 7(4):48, 2011.

[CEK09] Chandra Chekuri, Alina Ene, and Nitish Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, pages 42–55, 2009.

[CKS06] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing*, 2(7):137–146, 2006.

[CMS07] Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3(3), 2007.

[CR06] Shuchi Chawla and Tim Roughgarden. Single-source stochastic routing. In *Proceedings of APPROX*, pages 82–94. 2006.

[Dea05] Brian C. Dean. *Approximation Algorithms for Stochastic Scheduling Problems*. PhD thesis, MIT, 2005.

[DGV08] Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Math. Oper. Res.*, 33(4):945–964, 2008.

[FG15] Zachary Friggstad and Zhihan Gao. On Linear Programming Relaxations for Unsplittable Flow in Trees. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 265–283, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[GKR+03] Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, F. Bruce Shepherd, and Mihalis Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *J. Comput. Syst. Sci.*, 67(3):473–496, 2003.

[Kle98] Jon M. Kleinberg. Decision algorithms for unsplittable flow and the half-disjoint paths problem. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 530–539, 1998.

[KS04] Stavros G. Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using packing integer programs. *Math. Program.*, 99(1):63–87, 2004.

[KS06] Petr Kolman and Christian Scheideler. Improved bounds for the unsplittable flow problem. *J. Algorithms*, 61(1):20–44, 2006.

[Sch03] Alexander Schrijver. *Combinatorial optimization. Polyhedra and efficiency.*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003.

[Sri97] Aravind Srinivasan. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 416–425, 1997.