Approximation Algorithms for Throughput Maximization, Resource Minimization for Fire Containment and Hierarchical Clustering

by

Mirmahdi Rahgoshay

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

In this thesis, we present a variety of approximation algorithms for Resource Minimization for Fire Containment, Throughput Maximization, and Hierarchical Clustering.

Resource Minimization Fire Containment (RMFC) is a natural model for optimal inhibition of harmful spreading phenomena on a graph. In the RMFC problem on trees, we are given an undirected tree G, and a vertex r where the fire starts at, called root. At each time step, firefighters can protect up to B vertices of the graph while the fire spreads from burning vertices to all their neighbors that have not been protected so far. The task is to find the smallest B that allows for saving all the leaves of the tree. The problem is hard to approximate up to any factor better than 2 even on trees unless P $=$ NP [DM2010]. The main contribution to the RMFC problem is an asymptotic QPTAS for RMFC on trees. More specifically, let $\epsilon > 0$, and $\mathcal{I}$ be an instance of RMFC where the optimum number of firefighters to save all the leaves is $OPT(\mathcal{I})$. We present an algorithm which uses at most $\lceil (1+\epsilon)OPT(\mathcal{I}) \rceil$ many firefighters at each time step and runs in time $n^{O(\log\log n/\epsilon)}$. This suggests that the existence of an asymptotic PTAS is plausible especially since the exponent is $O(\log\log n)$. Our result combines a more refined height reduction lemma than the one in [SODA2019] with LP rounding and dynamic programming to find the solution. We also apply our height reduction lemma to the algorithm provided in [SODA2019] plus a more careful analysis to improve their 12-approximation and provide a polynomial time $(5+\epsilon)$-approximation.

In Throughput Maximization we have a collection J of n jobs, each having a release time r_j, deadline d_j, and processing time p_j. They have to be scheduled non-preemptively on m identical parallel machines. The goal is to find a schedule which maximizes the number of jobs scheduled entirely in their $[r_j, d_j]$ window. This problem has been studied extensively (even for the case of $m = 1$). Several special cases of the problem remain open. The approximability of the problem for $m = O(1)$ remains a major open question. We study the case of $m = O(1)$ and show that if there are c distinct processing times, i.e. p_j's come from a set of size c, then there is a randomized $(1-\varepsilon)$-approximation that runs in time $O(n^{mc^7\varepsilon^{-6}} \log T)$, where T is the largest deadline. Therefore, for constant m and constant c this yields a PTAS. Our algorithm is based on proving structural properties for a near optimum solution that allows one to use a dynamic programming with pruning.

In Hierarchical Clustering, one is given a set of n data points along with a notion of similarity/dis-similarity between them. More precisely for each two items i and j we are given a weight $w_{i,j}$ denoting their similarity/dis-similarity. The goal is to build a recursive (tree like) partitioning of the data points (items) into successively smaller clusters, which is represented by a rooted tree, where the leaves correspond to the items and each internal node corresponds to a cluster of all the items in the leaves of its subtree. Typically, the goal is to have the items that are relatively similar, to separate at deeper levels of the tree (and hence stay in the same cluster as deep as possible). Dasgupta [STOC2016] defined a cost function for a tree T to be $Cost(T) = \sum_{i,j\in[n]} \big(w_{i,j} \times |T_{i,j}|\big)$ where $T_{i,j}$ is the subtree rooted at the least common ancestor of i and j and presented the first approximation algorithm for such clustering. Later Cohen-Addad et al. [JACM2019] considered the same objective function as Dasgupta's but for dissimilarity-based metrics: $Rev(T) = \sum_{i,j\in[n]} \big(w_{i,j} \times |T_{i,j}|\big)$, where $w_{i,j}$ is the weight of dissimilarity between two nodes i and j. In this version a good clustering should have larger $T_{i,j}$ when $w_{i,j}$ is relatively large. It has been shown that both random partitioning and average linkage have ratio 2/3 which has been only slightly improved to 0.667078 [Charikar et al. SODA2020].

Our first main result is to improve this ratio of 0.667078 for $Rev(T)$. We achieve this by building upon the earlier work and use a more delicate algorithm and careful analysis which can be refined to achieve approximation 0.71604. We also introduce a new objective function for dissimilarity-based Hierarchical Clustering. Consider any tree T, we define $H_{i,j}$ as the number of i and j's common ancestors in T. In other words, when we think of the process of building tree as a top-down procedure, $H_{i,j}$ is the step in which i and j are separated into two clusters (they were stayed within the same cluster for $H_{i,j} - 1$ many steps). We suggest the cost of each tree T, which we want to minimize, to be $Cost_H(T) = \sum_{i,j\in[n]} \left(w_{i,j} \times H_{i,j}\right)$. We present a 1.3977-approximation for this objective.

Preface

I, Mirmahdi Rahgoshay, declare that this thesis titled, ‘Approximation Algorithms for Resource Minimization for Fire Containment, Throughput Maximization, and Hierarchical Clustering’ and the work presented in it are my own and mainly from three papers listed below:

- "Asymptotic Quasi-Polynomial Time Approximation Scheme for Resource Minimization for Fire Containment" (Conference version is accepted at STACS 2020 [68] and journal version is submitted to Algorithmica)
- "Approximations for Throughput Maximization" (Conference version is accepted at ISAAC 2020 [48] and journal version is submitted to Algorithmica)
- "Hierarchical Clustering: New Bounds and Objective" (Submitted to SODA 2022)

I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other quali

  cation at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.

To:

The great savior of the world

No two things have been combined better than knowledge and patience.

– Prophet Muhammad (peace be upon him).

Acknowledgements

There are many people I want to thank for both supporting me through my degree and for their help with this thesis. First and foremost, I want to thank my supervisor, Mohammad Reza Salavatipour. You have been a great teacher, mentor, and role-model who has pushed me to become a better student and researcher. While I have had my fair share of failures and stumbles along the way, you have shown me how much more I can achieve, and how much further I can still grow as a researcher.

I would like to thank Prof. Zachary Friggstad, Prof. Dale Schuurmans, Prof. Csaba Szepesvari, Prof. Nathan Sturtevant, Prof. Lorna Stewart and-Pro. Ben Moseley for accepting to be in my thesis committee.

I would also like to thank my beloved wife, Maryam Mahboub. Thank you for supporting me for everything, and especially I can't thank you enough for encouraging me throughout this experience.

Finally I thank my God, for helping me through all the difficulties. I have experienced Your guidance day by day. You are the one who let me finish my degree. I will keep on trusting You for my future. Thank you, Lord.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis we discuss some **NP**-hard optimization problems and try to design algorithms for them. Since it is very unlikely to find an exact algorithm for **NP**-hard problems with polynomial running time, we try to approximately solve them by providing approximation algorithms.

The first problem we consider is a special problem of a wide varity of Resource Allocation Problems. Resource Alocation has been a fundamental area in Computing Science over the past several decades. In Resource Minimization Fire Containment (RMFC) Problem we are trying to contain a harmful spreading phenomena on a graph, using limited resources to protect nodes.

The second problem we consider is a Scheduling Problem, namely Throughput Maximization. Scheduling jobs with arrival times and deadlines is a fundamental problem in numerous areas of computer science and has been studied in various fields, including Operations Research over the past several decades. In throughput Maximization the goal is to complete as many jobs as possible by their deadline.

The last problem is on one of fundamental topics in Computer Science: Clustering. Clustering has been used in a wide variety of application areas such as machine learning, data mining, pattern recognition, image analysis, bioinformatics, database systems, and information retrieval. A hierarchical clustering is a recursive partitioning of a data set into clusters of successively smaller size.

1.1 Preliminaries

We begin by formalizing the terminology we will use throughout this thesis. The definitions given here are adapted from [75], [79], and [78].

1.1.1 Graphs

A graph G is defined by its finite edge set $E(G) = \{e_1, e_2, ..., e_m\}$ and finite vertex set $V(G) = \{v_1, v_2, ..., v_n\}$, where each edge $e \in E(G)$ is an unordered pair of vertices in $V(G)$. To simplify notation, we may drop the parameters of V and E when the graph is clear from context, and instead denote G as the pair (V, E). We also consider *directed* graphs; in a directed graph G, each edge $e \in E(G)$ is an *ordered* pair of vertices. We use the same notation as for undirected graphs.

In an undirected graph for each edge $e = uv \in E(G)$, we say u and v are *adjacent* and e is *incident* to u and v. The *neighbours* of a vertex v are the vertices u such that u and v are *adjacent*; we denote this set as $N_G(v)$, or simply $N(v)$ when G is clear from context.

A *subgraph* of a graph G is a graph H, where H is obtained from G by *deleting* some edges and/or some vertices (and their incident edges) from G. We notate this relation as $H \subseteq G$, and may simply say that G *contains* H or H is *in* G. A *subgraph* $H \subseteq G$ is *spanning* if $V(H) = V(G)$.

Walk in Graph

A walk in graph G is a finite non-empty sequence $W = v_0e_1v_1e_2v_2...e_kv_k$, whose terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i. We say that W is a walk from v_0 to v_k. The vertices v_0 and v_k are called the origin and terminus of W, respectively, and $v_1, v_2, \ldots, v_{k-l}$ its internal vertices. The integer k is the length of W. A walk is closed if it has positive length and its origin and terminus are the same.

Path in Graph

A walk whose origin, terminus and internal vertices are distinct is a path. We sometimes use the term 'path' to denote a graph corresponding to a path.

Cycle in Graph

A closed walk whose origin and internal vertices are distinct is a cycle. Just as with paths we sometimes use the term 'cycle' to denote a graph corresponding to a cycle. A cycle of length k is called a $k - cycle$.

1.1.2 Approximation Algorithms

Decision Problems and NP

A decision problem is a problem that can be answered with either "yes" or "no". We view decision problems as languages. A language is a subset of binary strings over the alphabet $\{0, 1\}$. Language L corresponding to some decision problem is the set of all strings in L that encode "yes" instances to the problem.

A language $L \in \mathbf{NP}$ if there are polynomials p, q and a Turing machine M (called a verifier) such that for each string $x \in \{0, 1\}^*$, the following holds. If $x \in L$, then a certificate string y of length at most $p(|x|)$ must exist such that $M(x, y)$ accepts in at most $q(|x|)$ steps. Otherwise, for all strings y of length at most $p(|x|)$, $M(x, y)$ rejects in at most $q(|x|)$ steps. **NP** is therefore the class of all languages for which there are short and quickly verifiable yes-certificates.

L_1 and L_2 be two languages in **NP**. A language L_1 reduces to L_2 if there is a Turing machine that, given the string $x \in \{0, 1\}^*$, outputs a string y such that $y \in L_2$ if and only if $x \in L_1$, and does so in $poly(|x|)$ steps. A language L is **NP**-hard if for every language $L_0 \in \mathbf{NP}$, L_0 reduces to L. A language L is **NP**-complete if L is **NP**-hard and $L \in \mathbf{NP}$.

Optimization problems

An **NP**-optimization problem Π consists of:

- A set of valid *instances* D_Π, where we can determine if some instance $I \in D_\Pi$ in time polynomial in $|I|$. We assume all *instances* $I \in D_\Pi$ can be expressed as finite binary strings; this implies all numeric values could be integer or rational. The *size* of an instance I, written $|I|$, is the number of bits needed to express it.

- A set of feasible solutions $\mathcal{S}_\Pi(I)$ for each instance $I \in D_\Pi$, where we can determine if $S \in \mathcal{S}_\Pi(I)$ in time $poly(|I|)$. The length of each solution must be polynomially bounded in the length of I.

- An *objectivefunction* obj_Π that assigns each instance-solution pair (I, s) a non-negative value, computable in time that is polynomial in $|I|$.

We also specify whether Π is a a *minimization problem* or a *maximization problem*. For a minimization/maximization problem Π and instance $I \in D_\Pi$, an *optimal solution* is a feasible solution $s \in S_\pi(I)$ that minimizes/maximizes the value of obj_Π ; that is, $argmin_{s \in S_\Pi}\ obj_\Pi(I, s)$ or $argmax_{s \in S_\Pi}\ obj_\Pi(I, s)$, respectively. We denote such a solution as $OPT_\Pi(I)$, or simply OPT if the problem and instance are clear from context. We slightly abuse this notation by using OPT to also refer to the objective value of the optimum solution, when the type of OPT is clear from context.

An **NP** optimization problem Π gives rise to a class of **NP** decision problems Π', by asking if a feasible solution of at most/at least some objective value exists (for minimization/maximization problems, respectively). A polynomial time algorithm that solves Π can thus be used to answer the decision problem Π'. On the other hand, proving that the decision version of a problem $I' \in \Pi'$ is hard in some sense, shows that the optimization version $I \in \Pi$ is at least as hard as I'. For example if we prove that I' is **NP**-hard, then it means that I is **NP**-hard too.

Approximation algorithms

Let Π be a minimization (maximization) problem, and let $\alpha : \mathbb{Z}^+ \to \mathbb{Q}^+$ be a function such that $\alpha(|I|) \geq 1$ for all inputs $I \in D_\Pi$. An algorithm A is an $\alpha -$ *approximation* for Π if, for all instances I, A returns a feasible solution $S \in S_\Pi(I)$ such that $obj_\Pi(I, S) \leq \alpha(|I|).OPT_\Pi(I)$ $\left(obj_\Pi(I, S) \geq \frac{OPT_\Pi(I)}{\alpha(|I|)}\right)$ and the running time is bounded by $\mathbf{poly}(|I|)$. The function α is called the approximation ratio of A.

It is sometimes difficult to obtain an algorithm that meets this definition exactly. We might need to relax the running time bound, for example to a quasi-polynomial factor, which is $O(|I|^{\log^c(|I|)})$, where c is a constant. Or, the algorithm makes random choices, and so the approximation ratio only holds in expectation over all random choices. We still loosely refer to these as approximation algorithms, although we will state such relaxations explicitly.

An algorithm A is an approximation scheme for the minimization (maximization) problem Π if for the valid instance I and error parameter $\epsilon > 0$, it returns a feasible solution S such that $obj_{\Pi}(I,S) \leq (1+\epsilon).OPT_{\Pi}(I)$ $\big(obj_{\Pi}(I,S) \geq (1-\epsilon).OPT_{\Pi}(I)\big)$. We call A a *polynomial time approximation scheme* **PTAS** if its running time is **poly**$(|I|)$ for each fixed ϵ. We call A a *fully polynomial time approximation scheme* **FPTAS** if its running time is **poly**$(|I|, \frac{1}{\epsilon})$ for each fixed ϵ.

Problem Π is said to be in the class **PTAS** or **FPTAS** if it admits the respective approximation scheme. It is said to be in the class **APX** if it admits some constant approximation.

Let Π and Π' be two optimization problems. Π **PTAS**-reduces to Π' if there exists an algorithm A and function $c : \mathbb{R}^+ \rightarrow \mathbb{R}^+$, where for each valid instance I of Π and each fixed $\epsilon > 0$,

- Algorithm A returns an instance $I' = A(I, \epsilon)$ of Π' in time **poly**$(|I|)$, such that if I is feasible then I' is feasible, and
- Given any feasible solution $s' \in S_{\Pi}'(I')$, there exists a feasible solution $s \in S_{\Pi}(I)$ such that if $obj_{\Pi}'(I', s') \leq (1 + c(\epsilon)).OPT_{\Pi}'(I')$, then $obj_{\Pi}(I, s) \leq (1+\epsilon).OPT_{\Pi}(I)$

An optimization problem Π is said to be **APX**-hard if for every other problem $\Pi' \in$ **APX**, Π' **PTAS**-reduces to Π. If in addition $\Pi \in$ **APX**, then Π is said to be **APX**-complete.

Hardness of approximation

Roughly speaking, a *hardness* proof shows that a certain optimization problem cannot be approximated better than some threshold assuming certain complexity assumptions. As an extreme example, it was shown in [81] that the maximum independent set problem cannot be approximated better than $O(n^{1-\epsilon})$ for any constant $\epsilon > 0$ assuming $\mathbf{P} \neq \mathbf{NP}$, ruling out all but the most trivial approximations. A less extreme example, implied by the PCP theorem, is that approximating Max-3SAT better than $(1+\epsilon)$ for some $\epsilon > 0$ is **NP**-hard, ruling out a **PTAS** assuming $\mathbf{P} \neq \mathbf{NP}$ [75]. Since this problem is also **APX**-complete, a consequence of this hardness is that for any **APX**-hard optimization problem Π, $\Pi \notin$ **PTAS** unless **P** = **NP**.

SNP problems

In computational complexity theory, **SNP** (from Strict **NP**) is a complexity class containing a limited subset of **NP** based on its logical characterization in terms of graph-theoretical properties. It is defined as the class of problems that are properties of relational structures (such as graphs) expressible by a second-order logic formula of the following form:

$$\exists S_1 \ldots \exists S_n \forall v_1 \ldots \forall v_m \Phi(R_1, \ldots, R_k, S_1, \ldots, S_n, v_1, \ldots, v_m)$$

Where $R_1, \ldots, R_k$ are relations of the structure (such as the adjacency relation, for a graph), $S_1, \ldots, S_n$ are unknown relations (such as sets of tuples of vertices), and Φ is a any boolean combination of the relations with only existential second-order quantification over relations and only universal first-order quantification over vertices. For example, **SNP** contains 3-Coloring (the problem of determining whether a given graph is 3-colorable).

MAX-SNP problems

An analogous definition considers optimization problems, when instead of asking a formula to be satisfied for all tuples, one wants to maximize the number of tuples for which it is satisfied. That is, $MAX\text{-}SNP_0$ is defined as the class of optimization problems on relational structures expressible in the following form:

$$\max_{S_1,\ldots,S_n} |\{(v_1, \ldots, v_m) : \Phi(R_1, \ldots, R_k, S_1, \ldots, S_n, v_1, \ldots, v_m)\}|$$

$MAX\text{-}SNP$ is then defined as the class of all problems with a linear reduction to problems in $MAX\text{-}SNP_0$. For example, $MAX\text{-}3SAT$ is a problem in $MAX\text{-}SNP_0$: given an instance of 3-$CNF\text{-}SAT$ (the boolean satisfiability problem with the formula in conjunctive normal form and at most 3 literals per clause), find an assignment satisfying as many clauses as possible. Then $MAX\text{-}SNP$-hard problems are also defined just like **APX**-hard problems.

Unique Games Conjecture

In computational complexity theory, the unique games conjecture (often referred to as UGC) is a conjecture made by Subhash Khot in 2002 [54]. The conjecture postulates that the problem of determining the approximate value of a certain type of game, known as a unique game, is **NP**-hard. It has broad applications in the theory of hardness of approximation. If it is true, then for many important problems it is not only impossible to get an exact solution in polynomial time (as postulated by the **P** versus **NP** problem), but also impossible to get a good polynomial-time approximation.

However, *UGC* is not the only assumption to help for such an inapproximability results. For example it has been shown that the Minimum Vertex Cover problem is **NP**-hard to approximate to within a factor of $\sqrt{2}$, by assuming the traditional assumption of $\mathbf{P} \neq \mathbf{NP}$ [55], [56]

1.1.3 Linear Programming

Many problems in **NP** can be formulated as an *integer program* that describes the problem. Let $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$ be vectors, and $A = (a_{ij}) \in \mathbb{Q}^{m \times n}$ be a matrix. Let $u \cdot v$ denote the dot-product of two vectors u and v. The integer programming problem is to find a non-negative integer vector $x \in Z^{+n}$ maximizing the value $c \cdot x$, satisfying:

$$A \cdot x \leq b$$

Note that we can use this definition to define minimization problems as well (i.e. by maximizing $-c \cdot x$), and allow for $\geq$ and $=$ constraints.

Finding such a binary vector, or determining if such a vector even exists, is itself an **NP**-hard problem in general (otherwise, we could use integer programming to solve other **NP**-hard problems). Instead, suppose we relax this problem: instead of trying to find a binary vector x, we try to find a satisfying $x \in \mathbb{Q}^n$. This yields a *linear program*:

$$\begin{aligned} &\textbf{maximize} & c \cdot x & \qquad \textbf{(LP)} \\ &\textbf{subject to} & Ax \leq b, & \qquad (1.1) \\ & & x \geq 0 & \qquad (1.2) \end{aligned}$$

It is usually more convenient to explicitly write out the constraints and the objective function rather than specifying A, b, c directly, as in the following (equivalent) **LP**:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_j, \quad i = 1, ..., m \\ & \qquad x_j \geq 0, \quad j = 1, ..., n \end{aligned} \tag{1.3}$$

We say that we "solve" a linear program if we either determine no solution x exists, the value $c \cdot x$ is unbounded, or return a solution minimizing the objective c · x. Unlike integer programs, linear programs can be solved in time polynomial in n, m, and the number of bits Δ required to write the rational entries of A, b, and c; one such approach is the *interior point method* (see, for example, [53]).

Usefulness in approximations

Linear programming is a useful tool to build approximation algorithms with. The general procedure for a minimization (maximization) problem is to write the integer program, relax its constraints that force the variables to be non-negative integers, by allowing the variables to take any non-negative real numbers, solve it, and try to round the fractional result to an integer solution in polynomial time, without either violating constraints or increasing (decreasing) the objective value significantly. If we can do this while only increasing (decreasing) the objective value by a factor of $f(n)$, where $n = |x|$, then we will have an $f(n)$-approximation to the original problem.

1.2 Problems Considered and Main Results

1.2.1 Resource Minimization for Fire Containment

The Firefighter problem and a closely related problem named Resource Minimization Fire Containment (RMFC) are natural models for optimal inhibition of harmful spreading phenomena on a graph. The firefighter problem was formally introduced by Hartnell [45] and later Chalermsook and Chuzhoy [19] defined the RMFC problem. Since then, both problems have received a lot of attention in several research papers, even when the underlying graph is a spanning tree, which is one of the most-studied graph structures in this context.

In both problems (when restricted to trees) we are given a graph $G = (V, E)$, which is a spanning tree, and a vertex $r \in V$, called root. The problem is defined over discretized time steps. At time 0, a fire starts at r and spreads step by step to neighboring vertices. During each time step $1, 2, \ldots$ any non-burning vertex u can be protected, preventing u from burning in any future time step.

In the RMFC problem the task is to determine the smallest number $B \in \mathbb{Z}_{\geq 1}$ such that there is a protection strategy which protects B vertices at each time step while saving all the leaves from catching fire. In this context, B is referred to as the number of firefighters (or budget at each step). In the firefighters problem, given a fixed number of firefighters (i.e. number of vertices that can be protected at each time step) the goal is to find a strategy to maximize the number of vertices saved from catching the fire.

Main Results

By using Linear Programming and dynamic programming techniques, we show how to approximate RMFC with a small additive error by presenting a quasi-polynomial time asymptotic approximation scheme (**AQPTAS**) for it. More specifically our main result is the following theorem:

Theorem 1. *For RMFC on trees and for any $\epsilon > 0$ there is an algorithm that finds a solution using $\lceil (1+O(\epsilon))B \rceil$ firefighters with running time $n^{O(\log \log n/\epsilon)}$, where B is the optimal number of firefighters.*

We will also show how applying our more refined height reduction lemma to the algorithm used by Adjiashvili et al. [1], plus a more careful analysis, leads to a better constant factor. In particular, we obtain the following:

Theorem 2. *For any $\epsilon > 0$, there is a polynomial time $(5+\epsilon)$-approximation for the RMFC problem on trees.*

Recall that the RMFC problem on trees does not admit better than 2-approximation unless **P** = **NP** [57]. However, this does not rule out the possibility of a +1 approximation or an asymptotic **PTAS**. Our result is an indication that it is plausible that an asymptotic **PTAS** exists, especially since the exponent is $O(\log\log n)$, not $O(\log n)$ as we don't know any natural problem that admits $n^{O(\log\log n)}$ algorithm but not polynomial time.

We start by introducing a more powerful height reduction transformation than the one used in [1] that allows for transforming the RMFC problem into a more compact and better structured form, by only losing a $(1+\epsilon)$ factor in terms of approximability. This transformation allows us to identify small substructures, over which we can optimize efficiently, and having an optimal solution to these subproblems we can define a residual LP with small integrality gap. Then we will show how to apply dynamic programming on the transformed instance to obtain a strategy to protect the nodes at each step to successfully contain the fire and save all the leaves with using only $O(\epsilon B)$ more firefighters at each step. We will apply our more delicate height reduction lemma to the previous combinatorial approach [1] to reach a better constant factor approximation in polynomial time, which is presented in Theorem 2.

1.2.2 Throughput Maximization

Scheduling problems have been studied in various fields, including Operations Research and Computer Science over the past several decades. However, there are still several fundamental problems that are not resolved. In particular, for problems of scheduling of jobs with release times and deadlines in order to optimize some objective functions there are several problems left open (e.g. see [64, 70], [71]). We consider the classical problem of throughput maximization. In this problem, we are given a set J of n jobs where each job $j \in J$ has a processing time p_j, a release time r_j, as well as a deadline d_j. The jobs are to be scheduled non-preemptively on a single (or more generally on m identical) machine(s), which can process only one job at a time. The value of a schedule, also called its throughput, is the number of jobs that are scheduled entirely within their release time and deadline interval. Our goal is to find a schedule with maximum throughput.

Throughput maximization is a central problem in scheduling that has been studied extensively in various settings (even special cases of it are interesting open problems). They have numerous applications in practice [2, 39, 43, 59, 80]. The problem is known to be **NP**-hard (one of the list of problems in the classic book by Garey and Johnson [40]). In fact, even special cases of throughput maximization have attracted considerable attention. For the case of all p_j's being equal in the weighted setting (where each job has a weight and we want to maximize the total weight of scheduled jobs), the problem can be solved in polynomial time only when $m = O(1)$ (running time is exponential in m) [12, 34]. The complexity of the problem is open for general m. For the case where all processing times are bounded by a constant the complexity of the problem is listed as an open question [71]. It was shown in [35] that even for $m = 1$ and $p_j \in \{p, q\}$ where p and q are strictly greater than 1 the problem is **NP**-Complete.

Main Results

Our main result is the following. Suppose that there are c distinct processing times.

Theorem 3. *For the throughput maximization problem with m identical machines and c distinct processing times for jobs, for any $\varepsilon > 0$, there is a randomized algorithm which finds a $(1-\varepsilon)$-approximate solution with high probability runs in time $n^{O(mc^7\varepsilon^{-6})}\log T$, where T is the largest deadline.*

So for $m = O(1)$ and $c = O(1)$ we get a Polynomial Time Approximation Scheme (**PTAS**). Note that even for the case of $m = 1$ and $c = 2$, the complexity of the problem has been listed as an open problem in [71], however, it has been shown in [35] that even for $m = 1$ and $p_j \in \{p, q\}$ where p and q are strictly greater than 1 the problem is **NP**-Complete. Our algorithm for Theorem 3 is obtained by proving some structural properties for near optimum solutions and by describing a randomized hierarchical decomposition which allows us to do a dynamic programming.

An easy corollary of Theorem 3 is the following. If the largest processing time $p_{max} = \text{Poly}(n)$ then we get a quasi-polynomial time $(1-\epsilon)$-approximation using $(1+\epsilon)$-speed up of machines. This result of course was already obtained in [50]. We should mention that the framework of [50] heavily relies on machine speed up and it is not clear if that approach can be adapted to give an improved approximation for the original (non-augmented) machine speeds.

1.2.3 Hierarchical Clustering

Hierarchical Clustering has been studied and used extensively as a method for analysis of data. Suppose we are given a set of n data points (items) along with a notion of similarity between them. The goal is to build a hierarchy of clusters, where each level of hierarchy is a clustering of the data points that is a refined clustering of the previous level, and data points that are more similar stay together in deeper levels of hierarchy. In other words, we want to output a recursive partitioning of the items into successively smaller clusters, which are represented by a rooted tree, where the root corresponds to the set of all items, the leaves correspond to the items, and each internal node corresponds to the cluster of all the items in the leaves of its subtree. Many well-established methods have been used for Hierarchical Clustering, including some bottom-up agglomerative methods like single linkage, average linkage, and complete linkage, and some top-down approaches like the minimum bisection algorithm. In the bottom-up approaches, one starts from singleton clusters and at each step two clusters that are more similar are merged. For instance, in the average linkage, the average of pair-wise similarity of points in two clusters is computed and clusters which have the highest average are merged into one and this continues until one cluster (of all points) is created. In the top-down approaches, one starts with a single cluster (of all points) and each step a cluster is broken into two (or more) smaller ones. One such example is bisecting k-means (see [52]). Although these methods have been around for a long time, it was only recently that researchers tried to formalize the goal and objective of hierarchical clustering.

Suppose the set of data points of input are represented as the vertices of a weighted graph $G = (V, E)$ where for any two nodes i and j, $w_{i,j}$ is the weight (similarity or dissimilarity) between the two data points. Then one can think of a hierarchical clustering as a tree T whose leaves are nodes of G and each internal node corresponds to the subset of nodes of the leaves in that subtree (hence root of T corresponds to V). For any two data points i and j we use $T_{i,j}$ to denote the subtree rooted at the least common ancestor (LCA) of i and j and $w_{i,j}$ represents the similarity between i, j. In the very first attempt to define a reasonable objective function, Dasgupta [32] suggested the cost of each tree T, which we want to minimize, to be:

$$Cost(T) = \sum_{i,j\in[n]} \big(w_{i,j} \times |T_{i,j}|\big). \tag{1.4}$$

In other words, for each two items, i and j, the cost to separate them at a step is the product of their weight and the size of the cluster at the time we separate them. Intuitively, this means that the clusters deeper in the tree would contain items that are relatively more similar. Dasgupta [32] proved that the optimal tree must be a binary tree. He then analyzed this objective function on some canonical examples (such as complete graph) and proved that it is **NP**-hard to find the tree with the minimum cost. Finally, he showed that a simple top-down heuristic graph partitioning algorithm, namely using taking the (approximately) minimum sparsest cut, would have a provably good approximation ratio.

Cohen-Addad et al. [31] considered the same objective function but for dissimilarity-based graphs, where $w_{i,j}$ is the weight of dissimilarity between two nodes i and j. In this version a good clustering should have larger $T_{i,j}$ when $w_{i,j}$ is relatively large. Here the objective is to maximize the following formula:

$$Rev(T) = \sum_{i,j\in[n]} \big(w_{i,j} \times |T_{i,j}|\big) \tag{1.5}$$

They showed that the random top-down partitioning algorithm is a 2/3-

approximation and the classic average-linkage algorithm gives a factor 1/2 approximation (later [22] mentioned that the same analysis will show that it is actually $\frac{2}{3}$-approximation), and provided a simple top-down local search algorithm that gives a factor $(\frac{2}{3} - \epsilon)$-approximation. They take an axiomatic approach for defining 'good' objective functions for both similarity and dissimilarity-based hierarchical clustering by characterizing a set of admissible objective functions (that includes the one introduced by Dasgupta) that have the property that when the input admits a 'natural' ground-truth hierarchical clustering, the ground-truth clustering has an optimal value. They also provided a similar analysis showing that the algorithm of Dasgupta [32] (using sparsest cut algorithm) has ratio of $O(\alpha)$; their analysis of this is different (and slightly better) than [21]. More recently Chatziafratis et al. [24] showed that it is hard to approximate $Rev(T)$ within a constant of $9159/9189 = 0.996735$ assuming the Unique Games Conjecture.

Main Results

Our first result is to consider the revenue maximization of dissimilarity (i.e. objective (1.5)) and improve upon the algorithm of [22] which has ratio 0.667078:

Theorem 4. *For hierarchical clustering on dissimilarity-based graphs, there is an approximation algorithm to maximize objective of (1.5) with ratio* 0.71604.

To prove this, we build upon the work of [22] and present an algorithm that takes advantage of some conditions in which their algorithm fails to perform better. We start with a simpler algorithm which achieves ratio 0.6929. Then through a series of improvements we show how we can get to 0.71604 ratio.

Next we introduce a new objective function for hierarchical clustering and present approximation algorithms for this new objective. The intuition for this new objective function is that we expect the items that are more similar to remain in the same cluster for more steps, i.e. the step in which they are separated into different clusters is one which is further away from the root of the tree. Consider any tree T, we define $H_{i,j}$ as the number of common ancestors of i and j in T. In other words, when we think of the process of building the tree as a top-down procedure, $H_{i,j}$ is the step in which we decide to separate i and j from each other (they were decided to be together in $H_{i,j} - 1$ many steps). Intuitively, similar items are supposed to stay together until deeper nodes in the tree and hence are expected to have larger $H_{i,j}$ values whereas dissimilar items are supposed to be separated higher up in the tree. For instance, looking at a phylogenetic or genomic tree, the species that are most dissimilar are separated higher up in the tree (i.e. have small $H_{i,j}$) whereas similar species are separated at deep nodes of the tree and hence have high $H_{i,j}$ values. For dissimilarity-based graphs, we propose to minimize the following objective:

$$Cost_H(T) = \sum_{i,j \in [n]} \big(w_{i,j} \times H_{i,j}\big). \tag{1.6}$$

The problem we are looking to solve here is to find a full binary tree with the minimum $Cost_H(.)$. Our second main result is the following:

Theorem 5. *For hierarchical Clustering on dissimilarity-based graphs, a top-down algorithm that chooses the approximated weighted max-cut at each step, would be a $\frac{4\alpha_{GW}}{4\alpha_{GW}-1}$-approximation algorithm to minimize $Cost_H(T)$, where α_{GW} is the ratio of the max-cut approximation algorithm.*

Considering that the best known approximation algorithm for weighted maximum cut problem has ratio $\alpha_{GW} = 0.8786$ [41], the ratio of this algorithm would be 1.3977. This also means that any top-down algorithm which cuts at least half of the weight of the remaining edges, including the random partitioning algorithm, would have approximation ratio 2.

Chapter 2

The RMFC Problem

Recall that in RMFC problem (when restricted to trees) we are given a graph $G = (V, E)$, which is a spanning tree, and a vertex $r \in V$, called root. The problem is defined over discretized time steps. At time 0, a fire starts at r and spreads step by step to neighboring vertices. During each time step $1, 2, \ldots$ any non-burning vertex u can be protected, preventing u from burning in any future time step.

The task is to determine the smallest number $B \in \mathbb{Z}_{\geq 1}$ such that there is a protection strategy which protects B vertices at each time step while saving all the leaves from catching fire. In this context, B is referred to as the number of firefighters (or budget at each step). In the firefighters problem, given a fixed number of firefighters (i.e. number of vertices that can be protected at each time step) the goal is to find a strategy to maximize the number of vertices saved from catching the fire.

By using Linear Programming and dynamic programming techniques, we show how to approximate RMFC with a small additive error by presenting a quasi-polynomial time asymptotic approximation scheme (**AQPTAS**) and prove Theorem 1 which is basically saying that for RMFC on trees and for any $\epsilon > 0$ there is an algorithm that finds a solution using $\lceil \big(1 + O(\epsilon)\big) B \rceil$ firefighters with running time $n^{O(\log \log n / \epsilon)}$, where B is the optimal number of firefighters.

We will also show how applying our more refined height reduction lemma to the algorithm used by Adjiashvili et al. [1], plus a more careful analysis, leads to a better constant factor and prove Theorem 2 which is saying For any $\epsilon > 0$, there is a polynomial time $(5+\epsilon)$-approximation for the RMFC problem on trees.

2.1 Prior Work

For RMFC on trees, King and MacGillivray [57] showed that it is **NP**-hard to decide whether one firefighter is sufficient or not. This means that there is no (efficient) approximation algorithm with an approximation factor strictly better than 2, unless **P**=**NP**. On the positive side, Chalermsook and Chuzhoy [19] presented an $O(\log^* n)$-approximation[1] where n is the number of vertices. Their algorithm is based on a natural Linear Programming (LP) relaxation, which is a straightforward adaptation of the one previously used for the Firefighter problem on trees and essentially matches the integrality gap of the underlying LP (the integrality gap of the underlying LP is $\Theta(\log^* n)$ [19]). Recently, Adjiashvili et al. [1] presented a 12-approximation for RMFC, which is the first constant factor approximation for the problem. Their result is obtained through a combination of the known LPs with several new techniques, which allows for efficiently enumerating subsets of super-constant size of a good solution to obtain stronger LPs. They also present a **PTAS** for the firefighter problem.

The Firefighter problem and RMFC, both restricted to trees, are known to be computationally hard problems. More precisely, Finbow, King, MacGillivray and Rizzi [38] showed the **NP**-hardness for the Firefighter problem on trees even when the maximum degree is three.

Several approximation algorithms have been proposed for both of these problems. Hartnell and Li [44] proved that a natural greedy algorithm is a $\frac{1}{2}$-approximation for the Firefighter problem. Later, Cai, Verbin and Yang [17] improved this result to $1 - \frac{1}{e}$, using a natural LP relaxation and dependent randomized rounding. Then Anshelevich, Chakrabarty, Hate, and Swamy [6]

[1]Here $\log^* n$ denotes the minimum numberk of successive logs of base two that have to be nested such that $\underbrace{\log\log\ldots\log\log}_{k \text{ times}} n \leq 1$.

showed that the Firefighter problem on trees can be interpreted as a monotone submodular function maximization (SFM) problem subject to a partition matroid constraint. This observation yields another $(1-\frac{1}{e})$-approximation by using a recent $(1-\frac{1}{e})$-approximation for monotone SFM subject to a matroid constraint [18].

Chalermsook and Vaz [20] showed that, for any $\epsilon > 0$, the canonical LP used for the Firefighter problem on trees has an integrality gap of $1-\frac{1}{e}+\epsilon$. This generalized a previous result by Cai, Verbin and Yang [17]. When restricted to some tree topologies this factor $1-\frac{1}{e}$ was later improved (see [51]) but, for arbitrary trees, that was the best known approximation factor for a few years.

Recently, Adjiashvili, Baggio and Zenklusen [1] have filled the gap between previous approximation ratios and hardness results for the Firefighter problem. In particular, they present approximation ratios that nearly match the hardness results, thus showing that the Firefighter problem can be approximated to factors that are substantially better than the integrality gap of the natural LP. Their results are based on several new techniques, which may be of independent interest.

Assuming a variant of the Unique Games Conjecture (UGC), the RMFC problem in general graphs is hard to approximate within any constant factor, according to a recent work by Lee [58] which is based on a general method of converting an integrality gap instance to a length-control dictatorship test for variants of the s-t cut problem. For further results and related work we refer the reader to [1].

2.2 Preliminaries and Overview of the Algorithm

Recall that we are given a tree $G = (V, E)$ rooted at a vertex r, from which we assume the fire starts. We denote by $\Gamma \subseteq V$ the set of all leaves of the tree. Given an instance $\mathcal{I}$ for RMFC and an integer parameter $B \geq 1$, called the budget or the number of firefighters, at each time step we can "protect" up to B non-burning vertices. Such vertices are protected indefinitely. Our goal is to find the smallest B and a protection strategy such that all the leaves Γ are saved from catching the fire. Observe that we say a vertex u is protected, if we directly place a firefighter in u, and a vertex v is saved when the fire does not reach to u, because of protecting some u on the unique v-r path. This smallest value of B is denoted by $OPT(\mathcal{I})$.

Let $L \in \mathbb{Z}_{\geq 1}$ be the depth of the tree, i.e. the largest distance, in terms of the number of edges, between r and any other vertex in G. After at most L time steps, the fire spreading process will halt. For $\ell \in [L] := \{1, \ldots, L\}$, let $V_\ell \subseteq V$ be the set of all vertices of distance ℓ from r, which we call the ℓ-th level of the instance. We also use $V_{\leq \ell} = \cup_{k=1}^{\ell} V_k$, and we define $V_{\geq \ell}$, $V_{<\ell}$, and $V_{>\ell}$ in the same way. Moreover, for each $1 \leq \ell < L$ and each $u \in V_\ell$, $P_u \subseteq V_{\leq \ell} \setminus \{r\}$ denotes the set of all vertices on the unique u-r path except for the root r, and $T_u \subseteq V_{>\ell}$ denotes the subtree rooted at u, i.e. descendants of u.

2.2.1 Linear Programming Relaxation

We use the following (standard) Linear Programming (LP) relaxation for the problem that is used in both [19] and [1].

$$\begin{array}{lrcll} \min & & B & & \\ & x(P_u) & \geq & 1 & \forall u \in \Gamma \\ & x(V_{\leq \ell}) & \leq & B \cdot \ell & \forall \ell \in [L] \\ & x & \in & \mathbb{R}_{\geq 0}^{V \setminus \{r\}} & \end{array} \tag{2.1}$$

Here $x(U) := \sum_{u \in U} x(u)$ for any $U \subseteq V \setminus \{r\}$. Note that with $x \in \{0,1\}^{V \setminus \{r\}}$ and $B \in \mathbb{Z}_{\geq 0}$ we get an exact description of RMFC where x is the characteristic vector of the vertices to be protected and B is the budget. The first constraint enforces that for each leaf u, one vertex between u and r will be protected, which makes sure that the fire will not reach u. The second constraint ensures that the number of vertices protected after each time step is at most $B \cdot \ell$ and makes sure that we are using no more than B firefighters per time step (see [19] for more details). Note that (as mentioned in [19]), there is an optimal solution to RMFC that protects, with the firefighters available at time step ℓ, only the vertices in V_ℓ. Hence, we can change the above relaxation to one with the same optimal objective value by replacing the constraints $x(V_{\leq \ell}) \leq B \cdot \ell$ by the constraints $x(V_\ell) \leq B$ for all $\ell \in [L]$.

$$\begin{array}{rrcll} \min & & B & & \\ & x(P_u) & \geq & 1 & \forall u \in \Gamma \\ & x(V_\ell) & \leq & B & \forall \ell \in [L] \\ & x & \in & \mathbb{R}_{\geq 0}^{V \setminus \{r\}} & \end{array} \tag{2.2}$$

Throughout the chapter we use a lemma of [1] which basically says that any basic feasible solution of LP(2.2) (and also LP(2.1)) is sparse. This is proved for the polytope of the firefighters problem, which has the same LP constraints (just different objective function). Consider any basic feasible solution x to LP(2.2). One can partition $supp(x) = \{v \in V \setminus \{r\} : x(v) > 0\}$ into two parts: x-loose vertices and x-tight vertices. A vertex $v \in V \setminus \{r\}$ is x-loose or simply loose if $v \in supp(x)$ and $x(P_v) < 1$. All other vertices in $supp(x)$, which are not loose, will be x-tight or simply tight.

Lemma 1 (Lemma 3.1 in [1]). *Let x be a vertex solution to LP(2.2) for RMFC, then the number of x-loose vertices is at most L, the depth of the tree.*

We will use this property crucially in the design of our algorithm. Also, as noted in [1], we can work with a slightly more general version of the problem in which we have different numbers of budgets/firefighters at each time step: say $B_\ell = m_\ell B$ (for some $m_\ell \in \mathbb{Z}_{\geq 0}$) firefighters for each time step $\ell \in [L]$ while we are still minimizing B. Lemma 1 is valid for this generalization too.

2.2.2 Height Reduction

The technique of reducing the height of a tree at a small loss in cost (or approximation ratio) has been used in different settings and various problems (e.g. network design problems). For RMFC, Adjiashvili et al. [1] showed how one can reduce an instance of the problem to another instance where the height of the tree is only $O(\log n)$ at a loss of factor 2. In a sense, the tree will be compressed into a tree with only $O(\log n)$ levels. Here we introduce a more delicate version of that compression, which allows for transforming any instance to one on a tree with $O(\frac{\log n}{\epsilon})$ levels at a loss of $1+\epsilon$ in the approximation. Our compression is similar to that of [1] with an initial delay and ratio $1+\epsilon$. One key property we achieve with compression, is that we can later use techniques with running time exponential in the depth of the tree.

Suppose that the initial instance is a tree with L levels and each level ℓ has a budget B_ℓ. To compress the tree to a low height one, we will first do a sequence of what is called up-pushes. Each up-push acts on two levels $\ell_1, \ell_2 \in [L]$ with $\ell_1 < \ell_2$ of the tree, and moves the budget B_{ℓ_2} of level ℓ_2 up to ℓ_1. This means the new budget of level ℓ_1 will be $B_{\ell_1} + B_{\ell_2}$ and for level ℓ_2 it will be 0.

We will show that one can do a sequence of up-pushes such that: (i) the optimal objective value of the new instance is very close to the one of the original instance, and (ii) only $O(\log L/\epsilon)$ levels have non-zero budgets. Finally, 0-budget levels can easily be removed through a simple contraction operation, thus leading to a new instance with only $O(\log L/\epsilon)$ depth. The following theorem is a more powerful version of Theorem 4.1 in [1] with some improvements such as reducing the loss to only $1+\epsilon$ (instead of 2) and some differences in handling of the first levels.

Theorem 6. *Let $G = (V, E)$ be a rooted tree of depth L. Then for some constants $c, d > 0$ (that only depend on ϵ) we can construct efficiently a rooted tree $G' = (V', E')$ with $|V'| \leq |V|$ and depth $L' = O(\frac{\log L}{\epsilon})$, such that:*

(i) If the RMFC problem on G has a solution with budget $B \in \mathbb{Z}_{\geq 0}$ at each level, then the RMFC problem on G' has a solution with non-uniform budgets of $B_\ell = B$ for each level $\ell < c$, and a budget of $B_\ell = m_\ell \cdot B$ for each level $\ell \geq c$, where $m_\ell = \left(\lceil (1+\epsilon)^{(\ell-d+1)} \rceil - \lceil (1+\epsilon)^{(\ell-d)} \rceil\right)$.

(ii) Any solution to the RMFC problem on G', where each level $\ell < c$ has a budget of $B_\ell = B$ and each level $\ell \geq c$ has a budget of $B_\ell = m_\ell \cdot B$ can be transformed efficiently into an RMFC solution for G with budget $\lceil (1+2\epsilon)B \rceil$.

Proof. We start by describing the construction of $G' = (V', E')$ from G. We first change the budget assignment of the instance and then contract all 0-budgets levels.

We set i^* to be the smallest integer such that $(1+\epsilon)^{i^*} \geq \frac{2(1+\epsilon)}{\epsilon^2}$ and we let $c = \lceil (1+\epsilon)^{i^*} \rceil$. The set of levels $\mathcal{L}$ in which the transformed instance will have non-zero budget contains the first $c-1$ levels of G and all the levels $\ell \geq c$ of G such that $\ell = \lceil (1+\epsilon)^i \rceil$ for some $i^* \leq i \leq \frac{\log L}{\log(1+\epsilon)} = O(\log L/\epsilon)$:

$$\mathcal{L} = \left\{ 1 \leq \ell \leq L \quad | \quad \ell < c \quad or \quad \ell = \lceil (1+\epsilon)^i \rceil \text{ for some } i^* \leq i \leq \left\lfloor \frac{\log L}{\log(1+\epsilon)} \right\rfloor \right\}$$

For all other levels $\ell \notin \mathcal{L}$ we first do up-pushes. More precisely, the budget of these levels $\ell \in [L] \setminus \mathcal{L}$ will be assigned to the closest level in $\mathcal{L}$ that is above ℓ (has smaller index than ℓ). We then remove all 0-budget levels by contraction. For each vertex v in a level $\ell_i = \lceil (1+\epsilon)^i \rceil \geq c$ we will remove all vertices in the levels $\ell_i < \ell < \ell_{i+1} = \lceil (1+\epsilon)^{i+1} \rceil$ from its sub-tree and connect all the vertices in level ℓ_{i+1} of its sub-tree to v directly. This leads to a new tree G' with a new set of leaves. Since our goal is to save all the leaves in the original instance, for each vertex $v \in G'$ such that $v \in G$ has some leaves in its contracted sub-tree, we will mark v as a leaf in G' and simply delete all its remaining subtree.

This finishes our construction of $G' = (V', E')$ and it remains to show that both (i) and (ii) hold. Note that the levels in G' correspond to levels of G in $\mathcal{L}$: the first c levels of G' are the same as the first c levels of G; for each $\ell > c$, level ℓ in G' is level $\lceil (1+\epsilon)^{\ell-c+i^*} \rceil$ of G.

Here we want to determine what will be the budget of each level of G'. For each $\ell < c = \lceil (1+\epsilon)^{i^*} \rceil$, the level ℓ of G' is the same as the level ℓ of G and has the same budget $B_\ell = B$, because these levels are not involved in up-pushes. For $\ell = c$, all the budgets from level $\lceil (1+\epsilon)^{i^*} \rceil$ to $\lceil (1+\epsilon)^{i^*+1} \rceil - 1$ in G are up-pushed to this level. This means that the budget for level c in G' is $B_c = \left(\lceil (1+\epsilon)^{i^*+1} \rceil - \lceil (1+\epsilon)^{i^*} \rceil \right) \cdot B$. Now for each $i^* < i \leq \lfloor \frac{\log L}{\log(1+\epsilon)} \rfloor$, all the budgets from levels $\lceil (1+\epsilon)^i \rceil$ to $\lceil (1+\epsilon)^{i+1} \rceil - 1$ in G are up-pushed to level $\lceil (1+\epsilon)^i \rceil$, which becomes level $i - i^* + c$ in G'; this means that the budget for this level of G' will be $\lceil (1+\epsilon)^{i+1} \rceil - \lceil (1+\epsilon)^i \rceil$. Setting $\ell = i - i^* + c$ and $d = c - i^*$, the budget of level ℓ in G', is $B_\ell = (\lceil (1+\epsilon)^{\ell-d+1} \rceil - \lceil (1+\epsilon)^{\ell-d} \rceil) \cdot B$.

To prove (ii), we use the following lemma:

Lemma 2. *For any two consecutive levels $\ell \geq c$ and $\ell+1$ in G', the difference between m_ℓ and $m_{\ell+1}$ is relatively small. More precisely: $m_\ell(1+2\epsilon) \geq m_{\ell+1}$*

Proof. Based on the definition of m_ℓ and $m_{\ell+1}$ we have:

$$\begin{aligned} m_\ell &= \lceil (1+\epsilon)^{(\ell-d+1)} \rceil - \lceil (1+\epsilon)^{(\ell-d)} \rceil \geq (1+\epsilon)^{(\ell-d+1)} - (1+\epsilon)^{(\ell-d)} - 1 \\ \Rightarrow m_\ell(1+\epsilon) &\geq (1+\epsilon)^{(\ell-d+2)} - (1+\epsilon)^{(\ell-d+1)} - (1+\epsilon). \qquad (2.3) \end{aligned}$$

On the other hand:

$$\begin{aligned} m_{\ell+1} &= \lceil (1+\epsilon)^{(\ell-d+2)} \rceil - \lceil (1+\epsilon)^{(\ell-d+1)} \rceil \leq (1+\epsilon)^{(\ell-d+2)} - (1+\epsilon)^{(\ell-d+1)} + 1 \\ &\leq m_\ell(1+\epsilon) + 2 + \epsilon \qquad \text{using (2.3)} \qquad (2.4) \end{aligned}$$

Also by our choice of c, d and $i^* = c - d$ we can conclude that:

$$\begin{aligned} m_\ell &= \lceil (1+\epsilon)^{(\ell-d+1)} \rceil - \lceil (1+\epsilon)^{(\ell-d)} \rceil \\ &\geq (1+\epsilon)^{(\ell-d+1)} - (1+\epsilon)^{(\ell-d)} - 1 = \epsilon(1+\epsilon)^{(\ell-d)} - 1 \\ &\geq \epsilon(1+\epsilon)^{(c-d)} - 1 = \epsilon \cdot (1+\epsilon)^{i^*} - 1 \geq \epsilon \frac{2(1+\epsilon)}{\epsilon^2} - 1 \\ \Rightarrow \quad m_\ell &\geq \frac{2+\epsilon}{\epsilon} \Rightarrow \quad \epsilon m_\ell \geq 2 + \epsilon. \qquad (2.5) \end{aligned}$$

Combining (2.4) and (2.5) completes the proof. □

Corollary 1. *For each $\ell \geq c$ and each budget $B > 0$:*

$$m_{\ell+1} \cdot B \leq m_\ell \cdot \lceil (1+2\epsilon)B \rceil$$

Notice that in the constructed graph G' for each level $\ell \geq c$, we have $B_\ell = m_\ell \cdot B$. Now consider the instance of the problem on graph G with budget $\lceil (1+2\epsilon)B \rceil$ at each level. We will show that by doing some down-pushes on G (i.e. move the budget of each level to some level down) we can construct G' again where the budget of each level ℓ is $m_\ell \cdot B$, and this means that if G' has a solution with budget $m_\ell \cdot B$ in each level, then G has a solution with uniform budget $\lceil (1+2\epsilon)B \rceil$.

Like before the set of levels $\mathcal{L}$ with non-zero budgets will be the same. Instead of up-pushes, we will down-push the budget from all levels $\ell \notin \mathcal{L}$ to the closest level in $\mathcal{L}$ which is below ℓ (i.e has larger index than ℓ). We will also down-push budget $\lceil 2\epsilon B \rceil$ from each level $\ell < c$ to level $\ell = c$.

By doing the same contraction, for each level $\ell < c$ we will have $B_\ell = B$ and for each level $\ell > c$ we will have $B_\ell = m_{\ell-1} \cdot \lceil (1+2\epsilon)B \rceil$, which is greater than $m_\ell \cdot B$ based on the above lemma.

The only remaining level to consider is level $\ell = c$. For this level, by doing down-pushes, we will have budget $B_c = B + \lceil 2\epsilon B \rceil \cdot c$. Our claim is that this is not less than $m_c \cdot B$, which is equal to $(\lceil (1+\epsilon)c \rceil - c) \cdot B$ (based on the definition of m_c):

$$\begin{aligned} B_c &= B + \lceil 2\epsilon B \rceil \cdot c \\ &\geq B + 2\epsilon B \cdot c = (1 + 2\epsilon c) \cdot B \\ &\geq \lceil 2\epsilon c \rceil \cdot B = \lceil (1+2\epsilon)c - c \rceil \cdot B \\ &\geq (\lceil (1+\epsilon)c \rceil - c) \cdot B = m_c \cdot B. \end{aligned}$$

This will complete the proof of the theorem, because by considering these down-pushes, any solution to the RMFC problem on G', where level $\ell \geq c$ has a budget of $B_\ell = m_\ell \cdot B$ and level $\ell < c$ has a budget of $B_\ell = B$, can be transformed efficiently into an RMFC solution for G with budget $\lceil (1 + 2\epsilon)B \rceil$. $\square$

In the following we assume that the depth of the tree is not more than $\frac{\log n}{\log(1+\epsilon)} + \frac{2(1+\epsilon)}{\epsilon^2}$, so $L = O(\frac{\log n}{\epsilon})$. After finding a solution with budget B for a tree with this height, then we could apply the compression theorem and find a solution for the original tree by having $\lceil \epsilon B \rceil$ more firefighters at each level.

2.2.3 Overview of the Algorithm

Given an instance $\mathcal{I}$, our first step of the algorithm is to use Theorem 6 to reduce $\mathcal{I}$ to an instance $\mathcal{I}'$ with $L = O(\log n/\epsilon)$ levels. Note that when we use B to refer to *core* budget for instance $\mathcal{I}'$ we mean each level ℓ has budget $m_\ell \cdot B$ for $\ell \geq c$, and budget B for each level $\ell < c$. Also, by $OPT(\mathcal{I}')$ we mean the smallest value B such that $\mathcal{I}'$ has a feasible solution with core budget B as above. By Theorem 6, if we find a solution with *core* budget B for $\mathcal{I}'$ then it can be transformed to a solution for $\mathcal{I}$ with budget $\lceil(1+2\epsilon)B\rceil$. So we focus on the height reduced instance $\mathcal{I}'$ from now on. We present an algorithm such that if $B \geq OPT(\mathcal{I}')$ then it finds a feasible solution to $\mathcal{I}'$ with core budget at most $\lceil(1+\epsilon)B\rceil$. Then, using binary search, we find the smallest value of B_o (for B) for which the algorithm finds a feasible solution. This would give us a solution of budget at most $\lceil(1+\epsilon)OPT(\mathcal{I}')\rceil$, which in turn implies a solution for $\mathcal{I}$ of value at most $\lceil(1+O(\epsilon))OPT(\mathcal{I})\rceil$.

So let us assume we have guessed a value $OPT(\mathcal{I}') \leq B_o$. We consider LP(2.2) (with fixed $B = B_o$) for $\mathcal{I}'$ with guessed core budget B_o. Let x^* be a basic feasible solution to this instance. Using Lemma 1 we know that there are at most L loose vertices. As we will see, when B_o is relatively large, i.e. $B_o > \frac{L}{\epsilon}$, then we can easily find an integer solution using core budget at most $\lceil(1+\epsilon)B_o\rceil$ and this yields the desired bound for the original instance.

The difficult case is when B_o is small compared to L. The difficulty lies in deciding which vertices are to be protected by the optimum solution in the top h levels of the tree for some $h = O(\log\log n)$; as if one has this information then we can obtain a good approximation as in [1].

One way to do this would be to guess all the possible subsets of vertices that could be protected by the optimal solution in the first h levels of the tree, but this approach would have a running time far greater than ours. Still, we can solve the problem on instance $\mathcal{I}'$ in quasi-polynomial time using a bottom-up dynamic programming approach. More precisely, starting with the leaves and moving up to the root, we compute for each vertex $u \in V$ the following table. Consider a subset of the available budgets, which can be represented as a vector $q \in [B_1] \times ... \times [B_L]$. For each such vector q and node v, we want to know whether or not using budgets described by q for the subtree T_v (subtree rooted at v) allows for disconnecting v from all the leaves below it, i.e. saving all the leaves in T_v. Since $L = O(\log n/\epsilon)$ and the size of each budget B_ℓ is at most the number of vertices, the table size is $n^{O(\log n/\epsilon)})$. Moreover, it is easy to show that this table can be constructed bottom-up in quasi-polynomial time using an auxiliary table and another dynamic programming, to fill each cell of the table.

This approach would have the total running time of $n^{O(\log n/\epsilon)}$, because of the size of the table. In order to reduce the running time to $n^{O(\log\log n/\epsilon)}$, we would consider each budget vector value rounded up to the nearest power of $(1+\frac{\epsilon^2}{(\log n)^2})$. So, instead of $O(n^L) = n^{O(\log n/\epsilon)}$ many options for budget vectors q, we will have $O((\log n/\epsilon)^{3L}) = n^{O(\log\log n/\epsilon)}$ many options and we will show how by being more careful in our dynamic programming on these budget vectors we can still compute the table in time $n^{O(\log\log n/\epsilon)}$; this leads to an approximation scheme (instead of the exact algorithm) for the instance $\mathcal{I}'$.

2.3 Asymptotic Approximation Scheme

As mentioned above, first we use the height reduction as discussed in the previous section to reduce the given instance $\mathcal{I}$ to a new one $\mathcal{I}'$ with $L = O(\frac{\log n}{\epsilon})$ levels. We assume we have guessed a value $B_o \geq OPT(\mathcal{I}')$. Recall that, as in the statement of Theorem 6, for some constants c, d (depending on ϵ) the budget of each level $\ell < c$ is $B_\ell = B_o$ and for each level $\ell \geq c$ the budget is $B_\ell = m_\ell \cdot B_o$ where $m_\ell = \big(\lceil(1+\epsilon)^{(\ell-d+1)}\rceil - \lceil(1+\epsilon)^{(\ell-d)}\rceil\big)$.

We consider two cases: (I) when $B_o > \frac{L}{\epsilon}$, and (II) when $B_o \leq \frac{L}{\epsilon}$. For the first case we show how we can find a solution with core budget at most $\lceil(1+\epsilon)B_o\rceil$ by rounding the standard Linear Programming relaxation. For the second case we show how we can use a bottom-up dynamic programming approach to find a quasi-polynomial time approximation scheme.

2.3.1 Easy Case: $B_o > \frac{L}{\epsilon}$

In this case we consider LP(2.2) (with fixed $B = B_o$) for this instance. If x^* is a feasible solution to this LP and $B_o > \frac{L}{\epsilon}$ then we add $L \leq \lceil \epsilon B_o \rceil$ extra budget (i.e. number of firefighters) to the first level which is enough to protect all the *loose* vertices. Since by using Lemma 1 we know that there are at most L loose vertices and we can protect them all in the first step using L extra firefighters.

It remains to show that by using a budget of $m_\ell \cdot B_o$ at every level ℓ, for $c \leq \ell \leq L$, and B_o for $\ell < c$, we can protect all the tight vertices and so all the leaves would be saved, by adding only L many extra firefighters to only the first level.

Observe that for each tight vertex v, either $x(v) < 1$, then we would have a loose vertex in P_v, or $x(v) = 1$. In the first case v is already saved by protecting the loose vertices in the first step. If we only consider vertices with $x(v) = 1$, we can see that the solution is integral itself for these vertices. So we have rounded a fractional solution with $B_o > \frac{L}{\epsilon}$ to an integral one by using only $\lceil \epsilon B_o \rceil$ more firefighters just in the first level. In this case we find a feasible solution with core budget $B_o + \lceil \epsilon B_o \rceil$ in polynomial time.

2.3.2 When $B_o \leq \frac{L}{\epsilon}$

Recall that we have a budget of $B_\ell = B_o < L/\epsilon$ for each level $\ell < c$ and $B_\ell = m_\ell \cdot B_o \leq m_\ell \cdot \frac{L}{\epsilon}$ for each $c \leq \ell \leq L$. We denote by q^* the L-dimensional total budget vector that has $q^*[\ell] = B_\ell$ for each $1 \leq \ell \leq L$. Also for each L-dimensional vector $q \in [B_1] \times [B_2] \times ... \times [B_L]$, we denote by $Q(q)$ the set of all vectors q' such that $q' \leq q$. Suppose that $|Q(q^*)| = m$. We first describe a simpler (and easier to explain) dynamic programming with running time $n^{O(\log n/\epsilon)}$. Then we change it to decrease the running time and have our final approximation scheme with running time $n^{O(\log \log n/\epsilon)}$.

First Algorithm

Our dynamic program (DP) consists of two DP's: an outer (main) DP and an inner DP. In our main DP table A we have an entry for each vertex v and each vector $q \in Q(q^*)$. This entry, denoted by $A[v, q]$, will store whether using budgets described by q for levels of T_v allows for disconnecting v from all leaves below it or not.

More formally, if we assume $v \in V_\ell$, then $A[v, q]$ would be true if and only if there is a strategy for T_v such that (i) all the leaves in T_v are saved, and (ii) the budget for levels of T_v are given by vector q in indices $\ell + 1, \ldots, L$, i.e. $q[\ell+1]$ for the first level of T_v (direct children of v) , $q[\ell+2]$ for the second level, and so on.

We compute the entry $A[.,.]$ in a bottom up manner, computing $A[v, q]$

after we have computed the entries for children of v. To compute cell $A[v, q]$, we would use another auxiliary table B. Suppose v has k children $u_1, \ldots, u_k$ and assume that we have already calculated $A[u_j, q']$ for every $1 \leq j \leq k$ and all vectors $q' \in Q(q)$. Then we define a cell in our auxiliary table $B[v, q', j]$ for each $1 \leq j \leq k$ and $q' \in Q(q)$, where $B[v, q', j]$ is supposed to determine if the budget vector q' is enough for the union of subtrees rooted at $u_1, \ldots, u_j$ to save all the leaves in $T_{u_1} \cup \ldots T_{u_j}$ or not, where the total budgets for union of those subtrees are given by q'. We can compute $B[v, q', j]$ having computed $A[u_j, q'']$ and $B[v, q' - q'', j - 1]$ for all $q'' \in Q(q')$. This means that we can compute each cell $A[v, q]$ using auxiliary table B and internal DPs and the running time is $O(n^2 \cdot m^3)$. We need to find $A[r, q^*]$. If this cell is true, then we can save all the leaves of the tree using q^* as the budget vector for each level and if it is false, B_o would not be enough.

The problem is that m_ℓ could be large ($m_L = O(n)$) and so the options we have for the budget of each level is $O(n)$. Recall that we can have $B_o \leq \frac{L}{\epsilon}$ many choices for $q[\ell]$ when $\ell < c$ and $m_\ell \cdot \frac{L}{\epsilon}$ many options when $c \leq \ell \leq L$. Using the definition of the m_ℓ: $m_\ell = O(\epsilon(1+\epsilon)^{\ell-d})$, and so the total possible different budget vectors we could have is:

$$m = \Big(\frac{L}{\epsilon}\Big)^{c-1} \times \prod_{\ell=c}^{L} \Big(m_\ell \cdot \frac{L}{\epsilon}\Big) \quad = \quad \Big(\frac{L}{\epsilon}\Big)^{c-1} \times L^{L-c+1} \times \prod_{\ell=c}^{L} \Big((1+\epsilon)^{\ell-d}\Big) \quad = \quad O\Big((nL)^L\Big)$$

This means that the total running time will be $O(n^L) = n^{O(\log n/\epsilon)}$ and this is an exact algorithm to solve the RMFC problem on instance $\mathcal{I}'$.

Reducing Budget Possibilities

To reduce the running time, we only consider budget vectors where each entry of the vector is a power of $(1 + \big(\epsilon/\log n\big)^2)$. In this case we have at most $O\big(\log{(m_\ell \cdot L)} \times (\frac{\log n}{\epsilon})^2\big) = O(\log^3 n)$ many options for ℓth entry of q for each $c \leq \ell \leq L$, and so $m = O((\log n)^L) = n^{O(\log\log n/\epsilon)}$. Also, we have to show how we can compute the entries of the table in time $n^{O(\log\log n/\epsilon)}$ and why this would give a $(1+\epsilon)$-approximation of the solution. For each real x, let $RU(x)$ denote the value obtained by rounding up x to the nearest power of $(1 + (\epsilon/\log n)^2)$.

The main idea is that if for each vector q we round up each entry q_i to $RU(q_i)$ and denote the new vector by $RU(q)$ then if $A[v,q] = true$ then $A[v, RU(q)]$ is also true. So we only try to fill in entries of the table that correspond to vectors q where each entry is a power of $(1+(\epsilon/\log n)^2)$. We show this can be done in time $n^{O(\log\log n/\epsilon)}$ and the total loss in approximation is at most $1+\epsilon$ at the root of the tree.

From now on, we assume each vector q has entries that are powers of $(1+(\epsilon/\log n)^2)$; and recall that $Q(q)$ is the set of all such vectors q' such that $q \leq q'$ and assume we have already calculated $A[u_j, q']$ for every vector $q' \in Q(q)$ (again with all entries being powers of $(1+(\epsilon/\log n)^2)$).

If we try to compute $A[v,q]$ from $A[u_j,q']$'s the same way, we need to calculate $B[v,q',j]$ for each $1 \leq j \leq k$ and each time we round up the results of addition/subtractions (such as $q-q'$) to the nearest power of $(1+(\epsilon/\log n)^2)$.

Reducing Height of Inner Table

To compute cell $A[v,q]$ then this round-up operation could happen $k = O(n)$ times and the approximation loss blows up. Instead, we consider a hypothetical full binary tree with root v and leaves (at the lowest level) being $u_1, \ldots, u_k$; this tree will have height $O(\log k) = O(\log n)$. Then we define a cell in our auxiliary table for each internal node of this tree. See Figure 2.1 for an illustration.

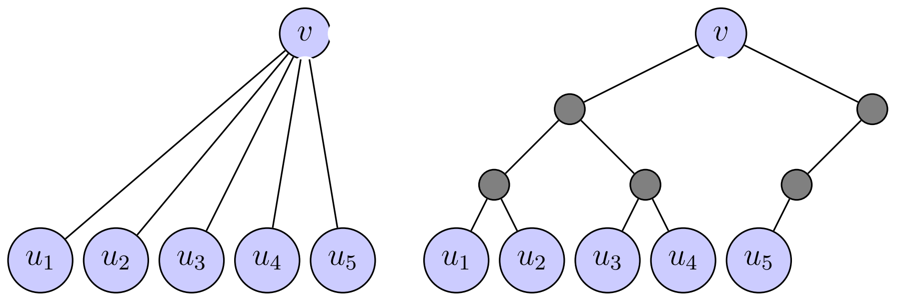


Figure 2.1: Illustration of the hypothetical full binary tree with root v and leaves $u_1, \ldots, u_5$

More formally we would define a cell in our auxiliary table $B[v,q',j,j']$ for each $0 \leq j \leq \lceil \log k \rceil$, $1 \leq j' \leq \lceil \frac{k}{2^j} \rceil$ and $q' \in Q(q)$ with all entries being powers of $(1+(\epsilon/\log n)^2)$, where $B[v,q',j,j']$ is supposed to determine if the budget vector q' is enough for the subtrees rooted at $u_{j_1}, \ldots, u_{j_2}$, where $j_1 = 2^j \cdot (j' - 1) + 1$ and $j_2 = min\{2^j \cdot j', k\}$, to save all the leaves in those subtrees, where the total budgets for the union of those subtrees is given by q'.

Similar to what we did before, we can compute $B[v, q', j, j']$ having computed $B[v, q'', j-1, 2j'-1]$ and $B[v, RU(q' - q''), j-1, 2j']$ (if it exists) for all $q'' \in Q(q')$. At each step we are computing a cell in table B a round-up will be applied to make the result of vector subtraction to be a vector with entries being powers of $(1 + (\epsilon/\log n)^2)$. If we can find a q'' such that both $B[v, q'', j-1, 2j'-1]$ and $B[v, RU(q' - q''), j-1, 2j']$ are true, then $B[v, q', j, j']$ would be true too. Also we can fill $A[v, q]$ by checking the value of $B[v, q_i, \lceil \log k \rceil, 1]$.

In the way we construct our auxiliary tables, while computing $A[v, q]$, when v has k children, $\log k$ many round up operations have happened (going up the auxiliary tree with root v) to the solution we found for T_v only in this step. This means that $O(\log k) \leq O(\log n)$ many round-ups could happen to compute entry $A[v, q]$ and the total number of round-ups starting from the values of $A[.,.]$ at a leaf level to $A[r, q]$ (for any q) would be at most $L \times \log n \leq \frac{\log^2 n}{\epsilon}$ and at each round-up we increase our budget by a factor of $(1 + (\epsilon/\log n)^2)$. So the total approximation increase while computing the entries for $A[r, .]$ would be at most:

$$\left(1 + \frac{\epsilon^2}{(\log n)^2}\right)^{\frac{\log^2 n}{\epsilon}} = 1 + O(\epsilon)$$

Observe that for every node v and subtree T_v if there is a solution with budget vectors q then there is a solution with budget vector $RU(q)$ as well. Using this fact we can find a solution with budget vector at most $(1 + O(\epsilon))q^*$ if there exists a solution with budget vector q^*. This completes the proof of Theorem 1.

2.4 Polynomial (5+ϵ)-Approximation for RMFC

In this section we show how the approach introduced in [1] can be adapted so that along with our height reduction lemma gives an algorithm with ratio

$(5+\epsilon)$ to prove Theorem 2.

We largely follow the proof of [1] only pointing out the main steps that need slight adjustments. We assume the reader is familiar with that proof and terminology used there.

Let x be a fractional solution to LP(2.2). We define W_x as the set of leaves that are (fractionally) cut off from r largely on low levels, i.e. there is high x-value on P_u on vertices far away from the root. We first start by recalling Theorem 4.2 from [1] which basically says that we can round an LP solution to an integral one by increasing the core budget B by a small constant such that W_x can be saved.

Theorem 7 (modified version of Theorem 4.2 in [1])**.** *Let $B \in \mathbb{R}_{\geq 1}$, $\mu \in (0,1]$, and $h = \lfloor \log_{1+\epsilon} L \rfloor$. Let $x \in$ LP(2.2) with value B and supp$(x) \subseteq V_{>h}$, and we define $W = \{u \in \Gamma | x(P_u) \geq \mu\}$. Then one can efficiently compute a set $R \subseteq V_{>h}$ such that:*

- *$R \cap P_u \neq \emptyset \quad \forall u \in W$, and*
- *There is an integral solution $z = y_1 + y_2$ to LP(2.2), which is a combination of two integral solutions y_1 and y_2 with value $B' = \frac{1}{\mu}B$ and 1 respectively such that supp$(y_1) \subseteq V_{>h}$ and supp$(y_2) \subseteq V_{\leq h}$.*

Proof. The proof would be very similar to the proof of Theorem 4.2 in [1], and the only difference is in providing the extra budget for protectecting the loose vertices in $V_{>h}$. They changed B to $B+1$ at level $h+1$ to provide this required budget. It that was enough, because the budget in the reduced instance is $B_{h+1} = 2^{h+1} \cdot B$ at this level, and so by this change $2^h = L$ many more firefighters are available and they are enough to protect all the loose vertices. But we need to change B to $B+1$ on all levels 1 to h, to have L many more firefighters for protecting all the loose vertices. This is because our budget in the reduced instance is $B_\ell = B$ when $1 \leq \ell < c$ and $B_\ell = m_\ell \cdot B$ when $c \leq \ell \leq L$. So by this change, we should have $c-1$ more firefighters in total for the first $c-1$ levels and $\sum_{\ell=c}^{h} m_\ell$ many more firefighters for levels c to h and the total would be $(1+\epsilon)^h = L$, which is enough to protect all the loose vertices. But the difference in our integral solution is that all the added budgets are from levels 1 to h (one for each level), and the remaining integral solution, which is $\frac{1}{\mu}$ feasible, is the subset of $V_{>h}$. This completes the proof of this theorem. $\square$

Similar to [1], we consider two cases based on how B compares to $\log L$. When $B \geq \log L$, we will have a 3-approximation for the reduced instance, by first solving the LP(2.2). This is similar to Theorem 4.3 in [1] and consistent with our height reduction lemma:

Theorem 8 (modified version of Theorem 4.3 in [1])**.** *There is an efficient algorithm that computes a feasible solution to a compressed instance of RMFC with budget at most $3B_{OPT}$ when $B_{OPT} \geq \log L$.*

Proof. The proof is largely following Theorem 4.3 in [1]. Here is a short version adapted. Assume x is a fractional LP(2.2) solution with value B. Then we use Theorem 7 and set $\mu = 1/2$ to obtain an integral solution z, which saves $W = \{u \in \Gamma | x(P_u) \geq \mu\}$, by core budget 1 at each level $1 \leq \ell \leq h$ and $2B$ at each level $h+1 \leq \ell \leq L$. Note that we can now transfer the 1 unit of budget from the very first level $\ell = 1$ to level $h+1$ and change the core budget $2B$ to $2B+1$ on this level and remove that extra budget from the very first level. This is because these extra firefighters from levels 1 to h are supposed to protect the loose vertices, which are in $V_{>h}$. By doing so we have an integral solution z such that the core budget is 0 in the first level, 1 in levels 2 to h, $2B+1$ at level $h+1$, and $2B$ at level $h+2$ to L. Now consider leaves $\Gamma \setminus W$. If we write another LP similar to LP(2.2), but specifically to save only these leaves by only protecting the vertices in $V_{\leq h}$, this LP would be feasible. Because all these vertices had $x(P_u) \cap V_{\leq h} \geq 0.5$, and so, $2x$ restricted to the vertices in $V_{\leq h}$, would be a feasible solution to this LP. Hence, we can find the optimal solution to this LP call it y. Based on Lemma 1, there would be at most $h = \log L$ many loose vertices all in $V_{\leq h}$, and so by adding $B > \log L = h$ many firefighters in the first level we would be able to protect all these y-loose vertices. Then all other remaining vertices could be saved by core budget $2B$. Putting these two solutions together (for saving W and $\Gamma \setminus W$) we have found an integral solution to save all the leaves, by having core budget $3B$ in the first level, $2B+1$ in levels 2 to $h+1$, and $2B$ at the remaining levels. This completes the proof of this theorem. $\square$

We say (A, D) is a clean pair compatible with OPT, if $A \cup D \subseteq V_{\leq h}$, $A \subseteq OPT$ and $D \cap OPT = \emptyset$, for $h = \log\log L$. We also define $LP(A, D)$ by adding two sets of constraints to LP(2.2) to force the solution to pick all vertices in A and not picking all vertices in D as well as the vertices in their path to the root. Also for each fractional solution to this LP let $W_x = \big\{u \in \Gamma \big| x(P_u \cap V_{>h}) \geq \frac{1}{1+\epsilon}\big\}$ to be the set of leaves cut off from the root by an x-load of at least $\mu = \frac{1}{1+\epsilon}$ within bottom levels (we changed $\frac{2}{3}$ to $1/(1+\epsilon)$ from [1]). For each $u \in \Gamma \setminus W_x$, let $f_u \in V_{\leq h}$ be the vertex closest to the root among all vertices in $(P_u \cap V_{\leq h}) \setminus D$, then define $F_x = \{f_u | u \in \Gamma \setminus W_x\} \setminus A$. It follows that no two vertices of F_x lie on the same leaf-root path. Furthermore, every leaf $u \in \Gamma \setminus W_x$ is part of the subtree T_f for precisely one $f \in F_x$. Also lets define $Q_x = V_{\leq h} \cap (\cup_{f \in F_x} T_f)$.

Now we are ready to provide our modification of Lemma 4.4 in [1] when $B < \log L$:

Lemma 3 (modified version of Lemma 4.4 in [1])**.** *Let (A, D) be a clean pair of vertices (A, D), which is compatible with OPT, and let x and y be optimal solutions to $LP(A, D)$ and $LP(A, V_{\leq h} \setminus A)$ with objective function B and $\hat{B}$ respectively. Then, if $OPT \cap Q_x = \emptyset$, we have $\hat{B} \leq (2+\epsilon)B_{OPT}$.*

Proof. The proof is similar to the proof of Lemma 4.4 in [1] and the first difference is that we changed $\frac{2}{3}$ to $\frac{1}{1+\epsilon}$ in the definition of W_x. First of all we can have a fractional solution that saves W_x with picking only vertices from $V_{>h}$. This is because $(1+\epsilon)x$ restricted to levels $h+1$ to L would save W_x. Now partition $\Gamma \setminus W_x$ into two groups. The leaves that OPT cut them from the root by protecting a vertex in $V_{\leq h}$, denote them by W_1, and W_2 are the leaves that OPT is cutting them in levels $h+1$ to L. By finding such (A, D), we are actually saving W_1. and for W_2 there is an integral solution with core budget B_{OPT}, which is restricted to levels $h+1$ to L. So the optimum solution to $LP(A, V_{\leq h} \setminus A)$ would not use more than $(1+\epsilon)B_{OPT} + B_{OPT}$ as the core budget in levels $h+1$ to L. This completes the first part of lemma. To round this fractional solution to an integral one which saves W_x and W_2 (note that W_1 is saved already by the choice of A and D), we use the same technique as Theorem 7.

We need to first find an integral solution restricted to levels $h_1 = \log L$ to L that saves the leaves with $y(P_u \cap V_{>h_1}) \geq \frac{1}{2(1+\epsilon)}$ by adding one core budget to levels 1 to h_1 and then write another LP restricted to levels h to h_1. Then we find another integral solution restricted to levels h to h_1 by adding another core budget to levels 1 to h that saves all the remaining leaves, which for sure has $y(P_u \cap V_{>h} \cap V_{\leq h_1}) \geq \frac{1}{2(1+\epsilon)}$. Finally we would have an integral solution with core budget $B_{OPT}+2$ for the first h levels, $2(2+\epsilon)B_{OPT}+1$ for levels $h+1$ to h_1 and $2(2+\epsilon)B_{OPT}$ for levels h_1 to L. This completes the proof of this lemma. $\square$

The only remaining thing is to show how we can find such (A, D) pair of vertices in polynomial time that follows in the exact same way of Lemma 4.5 in [1].

Lemma 4 (modified version of Lemma 4.5 in [1]). *if $B_{OPT} \leq \log L$, then we can find such triple (A, D, x) satisfying the conditions of Lemma 3 in polynomial time.*

Proof. We start with $A = D = \emptyset$ and create a polynomial size list of (A, D) pairs and prove that if $B_{OPT} \leq \log L$, then at least one of the pairs in the list is what we are looking for. Initially our list contains only pair $(\emptyset, \emptyset)$ with label zero. Then we set $\gamma = h \cdot (1+\epsilon)^{(h+1)} \cdot B_{OPT} = O(\log^2 L \cdot \log\log L)$ and for each pair (A, D) with label $\gamma' < \gamma$ in the list and for each $u \in F_x$ add two pairs $(A \cup \{u\}, D)$ and $(A, D \cup \{u\})$ to the list with label $\gamma' + 1$ to the end of the list.

Consider a vertex $f_u \in F_x$ where $u \in \Gamma \setminus W_x$, since u is a leaf outside W_x we have $x(P_u \cap V_{\leq h}) > 1 - \frac{1}{1+\epsilon} > \epsilon$. Thus $\forall f_u \in F_x$ we have $x(T_{f_u} \cap V_{\leq h}) > \epsilon$ which means $\epsilon|F_x| < \sum_{f \in F_x} x(T_f \cap V_{\leq h})$. Because no two vertices of F_x lie on the same leaf-root path, the sets $T_{f_u} \cap V_{\leq h}$ are all disjoint for different $f_u \in F_x$ and hence $\sum_{f\in F_x} x(T_f \cap V_{\leq h}) \leq x(V_{\leq h}) \leq \sum_{\ell=1}^{h}(1+\epsilon)^{\ell}B < \frac{(1+\epsilon)^{h+1}}{\epsilon}B$. Considering the assumption $B \leq \log L$, then we have $|F_x| < \frac{(1+\epsilon)\log^2 L}{\epsilon^2}$.

Notice that the number of different pairs in our list with label 1 is at most $2|F_x|$, and label 2 at most $(2|F_x|)^2$ and so on. So the total size of our list would be $O((2|F_x|)^{\gamma})$ which is polynomial considering $L \leq O(\log n)$.

So, it remains to show that at least one pair in our list is satisfying the conditions of Lemma 3. For any clean pair (A, D) compatible with OPT, we define a potential function $\Phi(A, D) \in Z_{\geq 0}$ in the following way. For each $u \in OPT \cap V_{\leq h}$, let $d_u \in Z_{\geq 0}$ be the distance of u to the first vertex in $A \cup D \cup \{r\}$ when following the unique u-r path. We define $\Phi(A, D) = \sum_{u\in OPT\cap V\leq h} d_u$. Notice that as long as we have a triple (A, D, x) on our execution path that does not satisfy the conditions of Lemma 3, then the next triple (A', D', x') on our execution path satisfies $\Phi(A', D') < \Phi(A, D)$. Hence, either we will encounter a triple on our execution path satisfying the conditions of Lemma 3 while still having a strictly positive potential, or we will encounter a triple (A, D, x) compatible with OPT and $\Phi(A, D) = 0$, which implies $OPT \cap V \leq h = A$ and we thus correctly guessed all vertices of $OPT \cap V_{\leq h}$ implying that the conditions of Lemma 3 are satisfied for the triple (A, D, x). Since $\Phi(A, D) \geq 0$ for any compatible clean pair (A, D), this implies that a triple satisfying the conditions of Lemma 3 will be encountered if the recursion depth γ is at least $\Phi(\emptyset, \emptyset)$. To evaluate $\Phi(\emptyset, \emptyset)$ we have to compute the sum of the distances of all vertices $u \in OPT \cap V_{\leq h}$ to the root. The distance of u to the root is at most h since $u \in V_{\leq h}$. Moreover, $|OPT \cap V_{\leq h}| < (1+\epsilon)^{(h+1)}B_{OPT}$ due to the budget constraints, implying that a triple fulfilling the conditions of Lemma 3 is encountered. $\square$

This means that we are able to find a $(5+\epsilon)$-approximation for the reduced instance of the RMFC problem to have a $(5+\epsilon)$-approximation.

Chapter 3

Throughput Maximization

Recall that in Throughput Maximization we are given a set J of n jobs where each job $j \in J$ has a processing time p_j, a release time r_j, as well as a deadline d_j. The jobs are to be scheduled non-preemptively on a single (or more generally on m identical) machine(s), which can process only one job at a time. The value of a schedule, also called its throughput, is the number of jobs that are scheduled entirely within their release time and deadline interval.

By using Linear Programming and dynamic programming techniques, we provide a Polynomial Time Approximation Scheme (**PTAS**) for $m = O(1)$ and $c = O(1)$ where we assume there are c distinct processing times. This will prove Theorem 3 which is saying For the throughput maximization problem with m identical machines and c distinct processing times for jobs, for any $\varepsilon > 0$, there is a randomized algorithm which finds a $(1-\varepsilon)$-approximate solution with high probability runs in time $n^{O(mc^7\varepsilon^{-6})} \log T$, where T is the largest deadline.

3.1 Prior Work

It appears the first approximation algorithms for this problem were given by Spieksma [72] where a simple greedy algorithm has shown to have approximation ratio 1/2. This algorithm will simply run the job with the least processing time between all the available jobs whenever a machine completes a job. He also showed that the integrality gap of a natural Linear Program relaxation is 2. Later on, Bar-Noy et al. [14] analyzed greedy algorithms for various settings and showed that for the case of m identical machines greedy algorithm has ratio $1-1/(1+1/m)^m$. This ratio is $1/2$ for $m=1$ and approaches $1-1/e$ as m grows.

In a subsequent work, Chuzhoy et al. [29] looked at a slightly different version, call it *discrete* version, where for each job j, we are explicitly given a collection $\mathcal{I}_j$ of intervals (possibly of different lengths) in which job j can be scheduled. A schedule is feasible if for each job j in the schedule, j is placed within one of the intervals of $\mathcal{I}_j$. This version (vs. the version defined earlier, which we call the "continuous" version) have similarities but none implies the other. In particular, the discrete version can model the continuous version if one defines each interval of size p_j of $[r_j, d_j]$ as an interval in $\mathcal{I}_j$. However, the number of intervals in $\mathcal{I}_j$ defined this way can be as big as $d_j - r_j + p_j$ which is not necessarily polynomial in the input size. Chuzhoy et al. [29] presented a $(1-1/e-\epsilon)$-approximation for the discrete version of the problem. Spieksma [72] showed that the discrete version of the problem is MAX-SNP hard using a reduction to a version of MAX-$3SAT$. No such approximation hardness result has been proved for the continuous version.

Berman and DasGupta [16] provided a better than 2 approximation for the case when all the jobs are relatively big compared to their window size. A pseudo-polynomial time exact algorithm for this case is presented by Chuzhoy et al. [29] with running time $O(n^{poly(k)}T^4)$, where $k = max_j(d_j - r_j)/p_j$ and $T = max_j d_j$.

For the weighted version of the problem, [11] showed that when we have uniform processing time $p_j = p$, the problem is solvable in polynomial time for $m = 1$. For $m = O(1)$ and with uniform processing time [12, 34] presented polynomial time algorithms. For general processing time 2-approximation algorithms are provided in [13, 16] and this ratio has been the best known bound for the weighted version of the problem. More recently, Im et al. [49] presented better approximations for throughput maximization for all values of m. For the unweighted case, for some absolute $\alpha_0 > 1 - 1/e$, for any $m = O(1)$ and for any $\epsilon > 0$ they presented an $(\alpha_0 - \epsilon)$-approximation in time $n^{O(m/\epsilon^5)}$. They also showed another algorithm with ratio $1 - O(\sqrt{(\log m)/m} - \epsilon)$ (for any $\epsilon > 0$) on m machines. This ratio approaches 1 as m grows. Furthermore, their $1-O(\sqrt{(\log m)/m}-\epsilon)$ ratio extends to the weighted case if $T = \text{Poly}(n)$.

Bansal et al. [10] looked at various scheduling problems and presented approximation algorithms with resource augmentation (a survey of the many resource augmentation results in scheduling is presented in [65]). An α-approximation with β-speed augmentation means a schedule in which the machines are β-times faster and the total profit is α times the profit of an optimum solution on original speed machines. In particular, for throughput maximization they presented a 24-speed 1-approximation, i.e. a schedule with optimum throughput however the schedule needs to be run on machines that are 24-times faster in order to meet the deadlines. This was later improved by Im et al. [50], where they developed a dynamic programming framework for non-preemptive scheduling problems. In particular for throughput maximization (in weighted setting) they present a quasi-polynomial time $(1-\epsilon, 1+\epsilon)$-bi-criteria approximation (i.e. an algorithm that finds a $(1-\epsilon)$-approximate solution using $(1+\epsilon)$ speed up in quasi-polynomial time). We should point out that the **PTAS** we present for c distinct processing time implies (as an easy corollary) a bi-criteria **QPTAS** as well, i.e. a $(1-\epsilon)$-approximation using $(1+\epsilon)$-speed up.

For the problem of machine minimization, where we have to find the minimum number of machines with which we can schedule all the jobs, the algorithm provided in [66] has approximation ratio $O(\sqrt{\log n/\log\log n})$ only when $OPT = \Omega(\sqrt{\log n/\log\log n})$, and ratio $O(1)$ when $OPT = \Omega(\log n)$. Later Chuzhoy et al. [27] presented an $O(OPT)$-approximation which is good for the instances with relatively small OPT. Combining this with the earlier works implies an $O(\sqrt{\log n/\log\log n})$-approximation. Chuzhoy and Naor [28] showed a hardness of $\Omega(\log\log n)$ for the machine minimization problem.

Another interesting generalization of the problem is when we assign a height to each job as well and allow them to share the machine as long as the total height of all the jobs running on a machine at the same time is no more than 1. The first approximation algorithm for this generalization is provided by [13] which has ratio 5. Chuzhoy et al. [29] improved it by providing an

$(e-1)/(2e-1) > 0.3873$-approximation algorithm which is only working for the unweighted and discrete version of the problem. The problem has also been considered in the online setting [15].

3.2 Preliminaries

Recall that we have a set J of n jobs where each job $j \in J$ has a processing time p_j, a release time r_j as well as a deadline d_j, we assume all these are integers in the range $[0, T]$ (we can think of T as the largest deadline). The jobs are to be scheduled non-preemptively on m machines which can process only one job at a time. We point out that we do not require T to be poly-bounded in n. For each job $j \in J$ we refer to $[r_j, d_j]$ as span of job j, denoted by $span_j$. We use OPT to denote an optimum schedule and opt the value of it. In the weighted case, each job j has a weight/profit w_j which we receive if we schedule the job within its span. The goal in throughput maximization is to find a feasible schedule with maximum weight of jobs. Like most of the previous works, we focus on the unit weight setting (so our goal is to find a schedule with maximum number of jobs scheduled).

We also assume that for each $p \in P$, all the jobs with processing time p in an optimum solution are scheduled based on earliest deadline first rule; which says that at any time when there are two jobs with the same processing time available the one with the earliest deadline would be scheduled. This is known as Jackson rules and we critically use it in our algorithms.

3.3 PTAS for constant c and m

For ease of exposition, we present the proof for the case of $m = 1$ (single machine) only and then extend it to the setting of multiple machines.

In order to prove Theorem 3 we use the following theorem as the base of our DP, which is a special case of the results achieved by Hyatt-Denesik [47] in his Master thesis:

Theorem 9 ([47]). *Suppose we are given B intervals over the time-line where* *the machines are pre-occupied and cannot be used to run any jobs, there are R distinct release times, D distinct deadlines, and m machines, where $R, D, B, m \in O(1)$. Then there is a* **PTAS** *for throughput maximization with time* $2^{\varepsilon^{-1}\log^{-4}(1/\varepsilon)} + \mathrm{Poly}(n)$.

3.3.1 Overview of the Algorithm

At a high level, the algorithm removes a number of jobs so that there is a structured near optimum solution. We show that the new instance has some structural properties that is amenable to a dynamic programming. At the lowest level of dynamic programming we have disjoint instances of the problem, each of which has a set of jobs with only a constant size set of release times and deadlines, with possibly a constant number of intervals of time being blocked from being used. For this setting we use the algorithm of Theorem 9. We start (at level zero) by breaking the interval $[0, T]$ into a constant q (where q will be dependent on ε) number of (almost) equal size intervals, with a random offset. Let us call these intervals $a_{0,1}, a_{0,2}, \ldots, a_{0,q}$. Assume each interval has size exactly T/q, except possibly the first and last (and for simplicity assume T is a power of q). For jobs whose span is relatively large, i.e. spans at least λ (where $\frac{1}{\varepsilon} \leq \lambda \leq \varepsilon q$) intervals, while their processing time is relatively small (much smaller than T/q), based on the random choice of break points for the intervals, we can assume the probability that the jobs position in the optimum solution is intersecting two intervals is very small. Hence, ignoring those jobs (at a small loss of optimum), we can assume that each of those jobs are scheduled (in a near optimum solution) entirely within one interval. For each of them we "guess" which of the λ intervals is the interval in which they are scheduled and pass down the job to an instance defined on that interval. For jobs whose span is very small (fits entirely within one interval), the random choice of the q intervals, implies that the probability of their span being "cut" by these intervals is very small (and again we can ignore those that have been cut by these break down). For medium size spans, we have to defer the decision making for a few iterations. We then try to solve each of the q instances, independently and recursively; i.e. we break the intervals again into roughly q equal size intervals and so on. If and when an instance generated has only $O(1)$ release times or deadlines we stop the recursion and use the algorithm of Theorem 9 to find a near optimum solution. So considering the hierarchical structure of this recursion, we have a tree with at most $O(\log_q T)$ depth and at most $O(n)$ leaves, which is polynomial in the input size. There are several technical details that one needs to overcome in this paradigm. One particular technical difficulty is for some jobs we decide to re-define their span to be a smaller subset of their original span by increasing their release time a little and decreasing their deadline a little. We call this procedure, cutting their "head" and "tail". This will be a key property in making our algorithm work. We will show (Lemma 9) that under some moderate conditions, the resulting instance still has a near optimum solution. This allows us to reduce the number of guesses we have to make in our dynamic program table and hence obtain Theorem 3. We should point out that the idea of changing the span or start/finish of a job was done in earlier works. However, using speed-up of machines one could "catch up" in a modified schedule with a near optimum one. The difficulty in our case is we do not have machine speed up.

3.3.2 Structure of a Near Optimum Solution

Consider an optimum solution OPT. One observation we use frequently is that such a solution is left-shifted, meaning that the start time of any job is either its release time or the finish time of another job. Therefore, we can partition the jobs in schedule OPT into continuous segments of jobs being run whose leftmost points are release times and the jobs in each segment are being run back to back. We call the set of possible rightmost points of these segments "slack times".

Definition 1. *(Slack times). Let slack times Ψ be the set of points t such that there is a release time r_i and a (possibly empty) subset of jobs $J' \subseteq J$, such that $t = r_i + \sum_{j \in J'} p_j$*

So the start time and finish time of each job in an optimum solution is a slack time. The following (simple) lemma bounds the size of Ψ

Lemma 5. *There are at most n^{c+1} different possible slack times, where c is the number of distinct processing times.*

Proof. We upper bound number of distinct $r_i + \sum_{j \in J'} p_j$ values. First note that there are only n different r_i values. Also, for each set $J' \subseteq J$, the sum $\sum_{j \in J'} p_j$ can have at most n^c possible values as the number of jobs in J' with a specific processing time can be at most n and we assumed there are only c distinct processing times. $\square$

Given error parameter $\varepsilon > 0$ we set $q = 1/\varepsilon^2$, $k = \log_q T$ and for simplicity of presentation suppose T is a power of q. We define a hierarchical set of partitions on interval $[0, T]$. For each $0 \le i \le k$, I_i is a partition of $[0, T]$ into $q^{i+1} + 1$ many intervals such that, except the first and the last intervals, all have length $\ell_i = T/q^{i+1}$, and the sum of the sizes of the first and last interval is equal to ℓ_i as well. We choose a universal random offset for the start point of the first interval. More precisely, we pick a random number $r_0 \in [0, \frac{T}{q}]$ and interval $[0, T]$ is partitioned into $q + 1$ intervals $I_0 = \{a_{0,0}, a_{0,1}, ..., a_{0,q}\}$, where $a_{0,0} = [0, r_0]$, and $a_{0,t} = [(t-1)\frac{T}{q} + r_0, t\frac{T}{q} + r_0]$ for $1 \le t \le q - 1$ and $a_{0,q} = [T - \frac{T}{q} + r_0, T]$. Note that the length of all intervals in I_0 is $\frac{T}{q}$, except the first and the last which have their length randomly chosen and the sum of their lengths is $\frac{T}{q}$.

Similarly each interval in I_0 will be partitioned into q many intervals to form partition I_1 with each interval in I_1 having length $\frac{T}{q^2}$ except the first interval obtained from breaking $a_{0,0}$ and the last interval in I_1 obtained from breaking $a_{0,q}$, which may be partitioned into less than q many, based on their lengths. All intervals in I_1 have size $\frac{T}{q^2}$ except the very first one and the very last one. We do this iteratively and break intervals of I_i (for each $i \ge 0$) into q equal sized intervals to obtain I_{i+1} (with the exception of the very first and the very last interval of I_{i+1} might have lengths smaller).

We set $\lambda = 1/\varepsilon = \varepsilon q$ and partition the jobs into classes $\mathfrak{J}_0, \mathfrak{J}_1, ..., \mathfrak{J}_k, \mathfrak{J}_{k+1}$, based on the size of their span. For each $1 \le i \le k$, job $j \in \mathfrak{J}_i$ if $\lambda \cdot \ell_i \le |span_j| < \lambda \cdot \ell_{i-1}$. Also $j \in \mathfrak{J}_0$ (and $j \in \mathfrak{J}_{k+1}$) if $\lambda \ell_0 \le |span_j|$ (and $|span_j| < \lambda \cdot \ell_k$). For each interval $a_{i,t}$ in level I_i, we denote the set of jobs whose span is entirely inside $a_{i,t}$ by $J(a_{i,t})$.

Based on our definitions of interval levels and job classes, we can say that for each $0 \le i \le k$ if $j \in \mathfrak{J}_i$, then $span_j$ would have intersection with at most $\lambda + 1$ (or fully spans at most $\lambda - 1$) many consecutive intervals from I_{i-1} and at least λ many consecutive intervals from I_i. Suppose $j \in I_i$ and $span_j$ has intersection with $a_{i,t_j}, a_{i,t_j+1}, ..., a_{i,t'_j}$ from I_i, then define $span_j \cap a_{i,t_j}$ and $span_j \cap a_{i,t'_j}$ as $head_j$ and $tail_j$, respectively.

We consider two classes of jobs as “bad” jobs and show that there is a near optimum solution without any bad jobs. The first class of bad jobs are those that we call “span-crossing”. For each job $j \in J$, we call it "span-crossing" if $j \in \mathfrak{J}_i$ for some $2 \le i \le k+1$ (so $\lambda \cdot \ell_i \le |span_j| < \lambda \cdot \ell_{i-1}$), and its span has intersection with more than one interval in I_{i-2}.

Lemma 6. *Based on the random choice of* r_0 *(while defining intervals), the expected number of span-crossing jobs in the optimum solution is at most* $\frac{\lambda+1}{q}\text{opt} = \mathrm{O}(\varepsilon\text{opt})$.

Proof. Observe that because $j \in \mathfrak{J}_i$, we have $|span_j| < \lambda \cdot \ell_{i-1}$. This means that the $span_j$ would have intersection with at most $\lambda + 1$ (or fully spans at most $\lambda - 1$) many consecutive intervals from I_{i-1}. Also because of the random offset while defining I_0, and since $\ell_{i-2} = q \cdot \ell_{i-1}$, the probability that job j being "span-crossing" will be at most $\frac{\lambda+1}{q}$. $\square$

So, we can assume with sufficiently high probability, that there is a $(1 - O(\varepsilon))$-approximate solution with no span-crossing jobs. The second group of bad jobs are defined based on their processing time and their position in the optimum solution. We then prove that by removing these type of jobs, the profit of the optimum solution will be decreased by a small factor. For each job $j \in J$, we call it "position-crossing" if $\ell_i \le p_j < \ell_{i-1}$ for some $2 \le i \le k+1$, and its position in OPT has intersection with more than one interval in I_{i-2}.

Lemma 7. *The expected number of position-crossing jobs in* OPT *is at most* $\frac{1}{q}\text{opt} = \mathrm{O}(\varepsilon^2\text{opt})$.

Proof. Consider OPT and suppose that $j \in J$ is a job with $\ell_i \leq p_j < \ell_{i-1}$. Observe that j can have intersection with at most 2 intervals in I_{i-2} because of its size. Considering our random offset to define interval levels, the probability of job j being a position-crossing (with respect to the random intervals defined) would be at most $\frac{1}{q}$ (since $p_j < \ell_{i-1} = \frac{\ell_{i-2}}{q}$). Thus, the expected number of position-crossing jobs in OPT is at most opt/q. $\square$

Hence, using Lemmas 6 and 7, with sufficiently high probability, there is a solution of value at least $(1 - O(\varepsilon))$opt without any span-crossing or position-crossing jobs. We call such a solution a canonical solution.

From now on, we suppose the original instance $\mathcal{I}$ is changed to $\mathcal{I}'$ after we first defined the intervals randomly and removed all the span-crossing jobs. So we focus (from now on) on finding a near optimum feasible solution to $\mathcal{I}'$ that has no position-crossing jobs. By OPT$'$ we mean such a solution of maximum value for $\mathcal{I}'$; we call that a canonical optimum solution. If we find a $(1 - O(\varepsilon))$-approximation to OPT$'$ (that has no position-crossing jobs), then using the above two lemmas we have a $(1 - O(\varepsilon))$-approximate solution to $\mathcal{I}$. So with OPT$'$ being an optimum solution to $\mathcal{I}'$ with no position-crossing jobs we let opt$'$ be its value.

3.3.3 Finding a Near Optimum Canonical Solution

As a starting point and warm-up, we consider the special case where instance $\mathcal{I}'$ only consists of jobs whose processing time is relatively big compared to their span and show how the problem could be solved. Consider the extreme case where for each $j \in J$, $p_j = |span_j|$. In this case the problem will be equivalent to the problem of finding a maximum independent set in an interval graphs which is solvable in polynomial time [42]. The following theorem shows that if $p_j \geq \frac{|span_j|}{\lambda}$ for each $j \in J$ (which we call them "tight" jobs), then we can find a good approximation as well. Therefore, it is the "loose" jobs (those whose processing time p_j is smaller than $\frac{|span_j|}{\lambda}$) that make the problem difficult. (we should point out that Chuzhoy et al. [29] also considered this special case and presented a DP algorithm with run time $O(n^{\text{Poly}(\lambda)}T^4)$ however, their DP table is indexed by integer points on the time-line and the polynomial dependence on T, which can be exponential in n, is unavoidable). The idea of the dynamic program of the next theorem is the basis of the more general case that we will prove later that handles "loose" and "tight" jobs together but the following theorem is easier to understand and follow and we present it as a warm-up for the main theorem.

Theorem 10. *If for all $j \in J$ in $\mathcal{I}'$, $p_j \geq \frac{|span_j|}{\lambda}$ then there is a dynamic programming algorithm that finds a canonical solution for instance $\mathcal{I}'$ with total profit* opt′ *in time $O(\varepsilon^{-1} n^{\varepsilon^{-2}c} \log T)$.*

Proof. Recall that $k = \log_q T$ and observe that for each $0 \leq i \leq k-1$ and each $j \in \mathcal{J}_i$: $\lambda \cdot \ell_i \leq |span_j| \leq \lambda p_j$, so $\ell_i \leq p_j$. Now if we somehow know $\mathrm{OPT}' \cap \mathcal{J}_0$ and $\mathrm{OPT}' \cap \mathcal{J}_1$ and remove the rest of jobs in $\mathcal{J}_0$ and $\mathcal{J}_1$, then the remaining jobs (which are all in $\mathcal{J}_{i \geq 2}$) have intersection with exactly one interval in I_0 (recall we have no span-crossing or position-crossing jobs), hence we would have $q+1$ many independent sub-problems (defined on the $q+1$ sub-intervals partitioned in level 0) with jobs from $\mathcal{J}_{i \geq 2}$.

So our first task is to "guess" the jobs in $\mathrm{OPT}' \cap (\mathcal{J}_0 \cup \mathcal{J}_1)$ (as well as their positions) and then remove the rest of the jobs in $\mathcal{J}_0 \cup \mathcal{J}_1$ from J as well as the jobs whose span is crossing any of the intervals in I_0; then recursively solve the problem on independent sub-problems obtained for each interval in I_0 together with the jobs whose spans are entirely within such interval. In order to guess the positions of jobs in $\mathrm{OPT}' \cap (\mathcal{J}_0 \cup \mathcal{J}_1)$ we use the fact that each job can start at a slack time. Since jobs in $\mathcal{J}_0 \cup \mathcal{J}_1$ have size at least $\ell_1 = T/q^2$, we can have at most q^2 of them in a solution. We guess a set S of size at most q^2 of such jobs and a schedule for them; there are at most $|\Psi|^{q^2} = n^{O(q^2 c)}$ choices for the schedule of S. Then we remove the rest of $\mathcal{J}_0$ and $\mathcal{J}_1$ from J for the rest of our dynamic programming. The guessed schedule of S defines a vector $\vec{v}$ of blocked spaces (those that are occupied by the jobs from S) and for each interval $a_{0,t}$, the projection of vector $\vec{v}$ in interval $a_{i,t}$, denote it by $\vec{v}_t$, has dimension at most q ($a_{0,t}$ has length $\ell_0 = T/q$ and each job in S has length at least $\ell_1 = T/q^2$). We pass each such vector $\vec{v}$ to the corresponding sub-problem.

Consider an interval $a_{i,t} \in I_i$ for some $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$. Recall that the set of jobs $j \in J$ whose span is completely inside $a_{i,t}$ is $J(a_{i,t})$. Because of the assumption of no span-crossing jobs, for each job $j \in J \setminus J(a_{i,t})$, if its span has intersection with $a_{i,t}$, then it would be in $\mathcal{J}_{i'}$ for some $i' \leq i+1$ (jobs from $\mathcal{J}_{i+2}$ are entirely within one interval of level I_i) and $|span_j|$ would be at least $\lambda\ell_{i+1}$, and hence $p_j \geq \ell_{i+1}$. Thus we can have at most $\ell_i/\ell_{i+1} = q$ such jobs. Assume we have a guessed vector $\vec{v}$ of length q where each entry of the vector denotes the start time as well as the end time of one of such jobs. This vector describes the sections of $a_{i,t}$ that are blocked for running such jobs from $J \setminus J(a_{i,t})$. The number of guesses for such vectors $\vec{v}$ is at most $n^{2q(c+1)}$ based on the bounds on the number of slack times. Given $\vec{v}$ and $J(a_{i,t})$ we want to schedule the jobs of $J(a_{i,t})$ in the free (unblocked by $\vec{v}$) sections of $a_{i,t}$.

Now we are ready to precisely define our dynamic programming table. For each $a_{i,t}$ and for each q-dimensional vector $\vec{v}$, we have an entry in our DP table A. This entry, denoted by $A[a_{i,t}, \vec{v}]$, will store the maximum throughput for an schedule of jobs running during interval $a_{i,t}$, using jobs in $J(a_{i,t})$ by considering the free slots defined by $\vec{v}$. The final solution would be $\max_S\{\sum_t A[a_{0,t}, \vec{v}_t] + |S|\}$, where the max is taken over all guesses S of jobs from $\mathcal{J}_0 \cup \mathcal{J}_1$ and $\vec{v}_t$ is the blocked area of $a_{i,t}$ based on S.

The base case is when $a_{i,t}$ has only constantly many release/deadline times. Given that we have also only constantly many processing times and $\vec{v}$ defines at most q many sections of blocked (used by bigger jobs) areas, then using Theorem 9 we can find a $(1 - O(\varepsilon))$-approximation in time Γ, where Γ is the running time of the **PTAS** for Theorem 9.

We can bound the size of the table as follows. First note that we do not really need to continue partitioning an interval $a_{i,t}$ if there are at most $O(1)$ many distinct release times and deadlines within that interval, since this will be a base case of our dynamic program. So the hierarchical decomposition of intervals $I_0, I_1, \ldots, I_k$ will actually stop at such an interval $a_{i,t}$ when there are at most $O(1)$ release times and deadlines. Therefore, at each level I_i of the random hierarchical decomposition, there are at most $O(n)$ intervals in I_i that will be decomposed into q more intervals in I_{i+1} (namely those that have at least a constant number of release times and deadlines within them). Thus the number of intervals at each level I_i is at most $O(nq)$ and the number of levels is at most $k = \log_q T$. Therefore, the total number of intervals in all partitions is bounded by $O(knq)$. To bound the size of the table A, each $\vec{v}$ has $n^{2q(c+1)}$ many options, based on the fact that we have at most n^{c+1} many choices of start time and end time (from the set Ψ of slacks) for each of the q dimensions of $\vec{v}$. Also as argued above, there are $O(knq)$ many intervals $a_{i,t}$ overall. So the size of table is at most $kqn^{O(qc)}$.

Now we describe how to fill the entries of the table. To fill $A[a_{i,t}, \vec{v}]$ for each $0 \leq i \leq k-1$ and $0 \leq t \leq \frac{T}{\ell_i}$, suppose $a_{i,t}$ is divided into q many equal size intervals $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ in I_{i+1}. We first guess a subset $\tilde{J_{i,t}}$ of jobs from $\mathcal{J}_{i+2} \cap J(a_{i,t})$, to be processed during interval $a_{i,t}$ consistent with free slots defined by $\vec{v}$. This defines a new vector $\vec{v}'$ that describes the areas blocked by jobs guessed recently as well as those blocked by $\vec{v}$. Projection of $\vec{v}'$ onto the q intervals $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ defines q new vectors $\vec{v}'_1, ..., \vec{v}'_q$. Now we check the sum of

$$A[a_{i+1,t'+1}, \vec{v}'_1] + A[a_{i+1,t'+2}, \vec{v}'_2] + ... + A[a_{i+1,t'+q}, \vec{v}'_q] + |\tilde{J_{i,t}}|$$

We would choose the $\tilde{J_{i,t}}$ which maximizes the above sum. Observe that jobs in $J(a_{i,t}) \setminus \mathcal{J}_{i+2}$ have length at most ℓ_{i+3} and because we have no position-crossing jobs, each of them is inside one of intervals $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ and would be considered in sub-problems.

Note that to fill each entry $A[a_{i,t}, \vec{v}]$ the number of jobs from $\mathcal{J}_{i+2}$ possible to be processed in $a_{i,t}$ would be at most q^2, because of their lengths. So the total number of guesses would be at most $n^{O(q^2 c)}$. This means that we can fill the whole table in time at most $kqn^{O(q^2 c)}$, where $q = 1/\varepsilon^2$ and $k = \log_q T$. $\square$

Considering Theorem 10, we next show how to handle "loose" jobs, i.e. those for which $p_j < \frac{|span_j|}{\lambda}$. Recall that for each $0 \leq i \leq k$ and for each $j \in \mathcal{J}_i$, if $span_j$ has intersection with intervals $a_{i,t_j}, a_{i,t_j+1}, ..., a_{i,t'_j}$ of I_i, then we denote $span_j \cap a_{i,t_j}$ and $span_j \cap a_{i,t'_j}$ as the head and tail of (span of) j, respectively.

Our next (technical) lemma states that if we reduce the span of each loose job by removing its head and tail then there is still a near optimum solution for $\mathcal{I}'$. More specifically, for each job loose $j \in \mathcal{J}_i$ ($p_j \leq \frac{|span_j|}{\lambda}$), whose span has intersection with intervals $a_{i,t_j}, a_{i,t_j+1}, ..., a_{i,t'_j}$ of I_i, we replace its release time to start at the beginning of a_{i,t_j+1} and its deadline to be end of a_{i,t'_j-1}; so $span_j$ will be replaced with with $span_j \setminus (a_{i,t_j} \cup a_{i,t'_j})$. Let this new instance be called $\mathcal{I}''$. Note that a feasible solution for instance $\mathcal{I}''$ would be still a valid solution for $\mathcal{I}'$ as well.

Lemma 8. *Starting from $\mathcal{I}'$, let $\mathcal{I}''$ be the instance obtained from removing the head and tail part of $span_j$ for each job $j \in J$ with $p_j \leq \frac{|span_j|}{\lambda}$. Then there is a canonical solution for $\mathcal{I}''$ with throughput at least $(1 - 120\varepsilon c)\mathrm{opt}'$.*

Proof. We will prove the following important key lemma in Section 3.4.

Lemma 9 (Head and tail cutting). *Consider any fixed processing time $p \in P$. Start with instance $\mathcal{I}'$ and remove only the head (or only the tail) part of $span_j$ for all jobs $j \in J$ with $p_j = p \leq \frac{|span_j|}{\lambda}$. Then there is a solution for the remaining instance with profit at least $(1 - \frac{60}{\lambda})\mathrm{opt}'$.*

Considering Lemma 9, the proof of Lemma 8 would be easy. We just need to apply Lemma 9 for all c many distinct processing times $p \in P$ and for both "head" and "tail". Then the total loss for removing all head and tail parts would be $\frac{60}{\lambda} \cdot 2c = 120\varepsilon$ fraction:

$$\mathrm{opt}(\mathcal{I}'') \geq (1 - \frac{60 \times 2\mathrm{c}}{\lambda})\mathrm{opt}' \geq (1 - 120\varepsilon\mathrm{c})\mathrm{opt}'.$$

□

The next theorem together with Lemmas 6, 7, and 9 will help us to complete the proof.

Theorem 11. *There is a dynamic programming algorithm that finds an optimum solution for instance $\mathcal{I}''$ in time $\varepsilon^{-3} n^{O(\varepsilon^{-6}c)} \log T$.*

Before presenting the proof of this theorem we show how this can be used to prove Theorem 3 for $m = 1$.

Proof of Theorem 3. Starting from instance $\mathcal{I}$ we first reduced it to instance $\mathcal{I}'$ at a loss of $1 - O(\varepsilon)$. Then remove the head and tail part of the span for all the loose jobs to obtain instance $\mathcal{I}''$. Based on Lemma 9, we only loose a factor of $(1 - O(\varepsilon c))$ compared to optimum of $\mathcal{I}'$. Theorem 11 shows we can actually find an optimum canonical solution to instance $\mathcal{I}''$. This solution will have value at least $(1 - O(\varepsilon c))$opt using Lemmas 6, 7, and 9. To get a $(1 - \varepsilon')$-approximation we set $\varepsilon' = \varepsilon/c$ in Theorem 11. The run time will be $c^3\varepsilon^{-3}n^{O(\varepsilon'^{-6}c^7)}\log T$. $\square$

Now we prove Theorem 11.

Proof. The idea of the proof is similar to that of Theorem 10. However, the presence of "loose" jobs needs to be handled too. Suppose $j \in \mathcal{J}_i$ is a loose job, so $\lambda\ell_i \leq |span_j| < \lambda\ell_{i-1}$ and $p_j \leq \frac{span_j}{\lambda} < \ell_{i-1}$. We break these loose jobs into two categories. For the loose jobs that $p_j < \ell_{i+1}$, because they are not position-crossing, their position in the final solution will have intersection with at most one interval of I_i (and so we can pass them down to lower sub-problems). But for loose jobs where $\ell_{i+1} \leq p_j < \ell_{i-1}$ we need to guess them (similar to the tight jobs) and we can do the guessing since their size (relative to ℓ_i) is big. In order to handle these guesses, we add one more vector to the DP table, and we do the guess for two consecutive levels of our decomposition as we go down the DP.

Suppose $P = \{p_1, p_2, ..., p_c\}$. For each interval $a_{i,t}$ ($0 \leq i \leq k$, $0 \leq t \leq \frac{T}{\ell_i}$), q^2-dimensional vector $\vec{v}$ (where $0 \leq v_i \leq n$), (qc)-dimensional vector $\vec{u} = (u_{1,1}, \ldots, u_{q,c})$, where each $u_{\gamma,\sigma}$, $0 \leq u_{\gamma,\sigma} \leq n$, we have an entry in our DP table A. Suppose $a_{i,t}$ is partitioned into intervals $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ in I_{i+1}. Entry $A[a_{i,t}, \vec{v}, \vec{u}]$, will store the maximum throughput of a schedule in interval $a_{i,t}$ by selecting subsets of jobs from the following two collections of jobs:

- $J(a_{i,t}) \cap \mathcal{J}_{\geq(i+2)}$
- $u_{\gamma,\sigma}$ many jobs with processing time p_σ where $p_\sigma < \ell_{i+2}$ whose span is the entire interval $a_{i+1,t'+\gamma}$, for each $1 \leq \gamma \leq q$, and $1 \leq \sigma \leq c$.

by considering the free slots defined by vector $\vec{v}$ (that describes blocked spaces by jobs of higher levels).

Vector $\vec{u}$ is defining the sets of jobs from loose jobs (from higher levels of DP table) whose span was initially much larger than ℓ_{i+1}, the guesses we made requires them to be scheduled in interval $a_{i+1,t'+\gamma}$ (of length ℓ_{i+1}) and hence their span is the entire interval $a_{i+1,t'+\gamma}$. Like before, $\vec{v}$ is defining the portions of the interval which are already used by bigger jobs (that are guessed at the higher levels), and for similar reasons as in Theorem 10, we only need to consider $\vec{v}$'s of size at most q^2 and each job listed in $\vec{v}$ will be denoted by its start position and end position (so there is $O(|\Psi|^{2q^2}) = n^{O(q^2c)}$ possible values for $\vec{v}$).

Similar to Theorem 10, suppose we start at I_0. We guess a subset of tight jobs from $\mathcal{J}_0$ to decide on their schedule. Note that tight jobs will have $p_j \geq \ell_0$. We also need to guess (and decide on their schedule) those "loose" jobs $j \in \mathcal{J}_0$ where $p_j \geq \ell_2 = T/q^3$ (since their position may cross more than one I_1 intervals in the final solution). So we guess a set $S_0 \subseteq \mathcal{J}_0$ with $|S_0| \leq q^3$ of jobs j where $p_j \geq \ell_2$ and a feasible schedule for them. This will take care of guessing tight and those loose jobs of $\mathcal{J}_0$ with $p_j \geq \ell_2$. We need to do similarly for jobs from $\mathcal{J}_1$, i.e. we need to guess a set of tight jobs j from $\mathcal{J}_1$ (note that for them $p_j \geq \ell_1$) and also guess (and decide on their schedule) those "loose" jobs $j \in \mathcal{J}_1$ with $p_j \geq \ell_2$. To do so, we guess a set $S_1 \subseteq \mathcal{J}_1$ of jobs j where $p_j \geq \ell_2 = T/q^3$ and a feasible schedule for them (given the guesses for S_0); note that $|S_0 \cup S_1| \leq q^3$ (since all of $S_0 \cup S_1$ must fit in $[0, T]$). For each such guess, their schedule projects a vector of blocked spaces (occupied time of machine). This will be vector $\vec{v}$. The projection of $\vec{v}$ to each interval $a_{0,t}$ will be $\vec{v}_t$ which is the blocked area of $a_{0,t}$. Note that although $\vec{v}$ has up to q^3 blocks, each $a_{0,t}$ can have at most q^2 blocks since each block has size at least $\ell_2 = T/q^3$ and each $a_{0,t}$ has size $\ell_0 = T/q$.

For all the other jobs in $\mathcal{J}_0 \cup \mathcal{J}_1$ that have $p_j < \ell_2$, because they are not position-crossing, we can assume their position (in the final solution) has intersection with only one interval of I_1. For all these jobs of $\mathcal{J}_0 \cup \mathcal{J}_1$, we use the assumption that there is a near optimum solution in which they are not scheduled in their head or tail. So for the jobs in $\mathfrak{J}_0 \cup \mathfrak{J}_1$ with processing time less than ℓ_2 we can re-define their span to a guessed interval of I_1; these guesses define the qc-dimensional vectors $\vec{u}_t$ for each of the q sub-intervals of $a_{0,t}$ at level I_1 (how many loose jobs from $\mathfrak{J}_0 \cup \mathfrak{J}_1$ with $p_j < \ell_2$ have their span redefined to be one of sub-intervals of $a_{0,t}$). The final solution will be $\max_{S_0,S_1}\{\sum_t A[a_{0,t}, \vec{v}_t, \vec{u}_t] + |S_0 \cup S_1|\}$, where the max is taken over all guesses $S_0 \subseteq \mathfrak{J}_0$, $S_1 \subseteq \mathfrak{J}_1$ and $\vec{u}_t$ as described above.

To bound the size of the table, as argued before, we would have at most $O(knq)$ many intervals in all of $I_0, I_1, \ldots, I_k$. For each of them we consider a table entry for at most $n^{O(q^2c)}$ many vectors $\vec{v}$, $n^{O(qc)}$ many vectors $\vec{u}$. So the total size of the table would be $(kq)n^{O(q^2c)}$.

Like before, the base case is when interval $a_{i,t}$ has $O(1)$ many release times and deadlines. These base cases $A[a_{i,t}, \vec{v}, \vec{u}]$ can be solved using Theorem 9 for each vector $\vec{v}$ and $\vec{u}$.

To fill $A[a_{i,t}, \vec{v}, \vec{u}]$ in general (when $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$ and there are more than $O(1)$ many release times and deadlines in $a_{i,t}$), suppose $a_{i,t}$ is divided into q many equal size intervals $a_{i+1,t'+1}, \ldots, a_{i+1,t'+q}$ in I_{i+1}. What we decide at this level is:

- make a decision for all the jobs $j \in \mathfrak{J}_{i+2} \cap J(a_{i,t})$; those that are bigger than ℓ_{i+3} will be scheduled or dropped by making a guess; the rest we narrow down their span (guess) to be one of the lower level sub-intervals of $a_{i,t}$ and will be passed down as $\vec{u}'$ to sub-problems below $a_{i,t}$;

- make a decision for jobs in $\vec{u}$: those that are bigger than ℓ_{i+3} will be scheduled or dropped; the rest we narrow down their span (by a guess) to be one of the lower level sub-intervals of $a_{i,t}$

As in the case of I_0, we need to guess a set of tight jobs from $\mathfrak{J}_{i+2} \cap J(a_{i,t})$ and some loose jobs j with $p_j \geq \ell_{i+3}$ and their positions to be processed in $a_{i,t}$ (considering the blocked areas defined by $\vec{v}$). Let S_0 with $s_0 = |S_0|$ be this guessed set. Note that $s_0 \leq q^3$ since $p_j \geq \ell_{i+3} = \ell_i/q^3$. Also for each non-zero $u_{\gamma,\sigma}$ where $p_\sigma \geq \ell_{i+3}$ we guess how many of those $u_{\gamma,\sigma}$ many jobs should be scheduled and where exactly in $a_{i+1,t'+\gamma}$ (consistent with $\vec{v}$ and S_0); let S_1 be this guessed subset and $|S_1| = s_1$. Note that $s_0 + s_1 \leq q^3$ and there are at most $|\Psi|^{2q^3}$ possible guesses for S_0 and S_1 together with their positions; thus a total of $n^{O(q^3c)}$ possible ways to guess $S_0 \cup S_1$ and guess their locations in the schedule. Then for each possible pair of such guessed sets S_0, S_1 we compute the resulting $\vec{v}'$; this defines the space available for the rest of the jobs in $J(a_{i,t}) \cap \mathcal{J}_{\geq i+3}$, and those defined by $\vec{u}$ where $p_j < \ell_{i+3}$ after blocking the space defined by $\vec{v}$ and the space occupied by the pair of guessed sets S_0, S_1 above. We divide $\vec{v}'$ into q many vectors $\vec{v}'_1, ..., \vec{v}'_q$, (as we divided $a_{i,t}$ into q intervals).

We also change $\vec{u}$ to $\vec{u}'$ by setting all the entries of $u_{\gamma,\sigma}$ with $p_\sigma \geq \ell_{i+3}$ to zero and guess how to distribute $\vec{u}'$ into q many (qc)-dimensional vectors $\vec{u}'_1, ..., \vec{u}'_q$ such that $\vec{u}'_1 + \vec{u}'_2 + ... + \vec{u}'_q = \vec{u}'$, where $\vec{u}'_\gamma$ is describing the number of jobs of different sizes whose span is re-defined to be one of the sub-intervals of $a_{i+1,t'+\gamma}$ at level I_{i+2}. The number of ways to break $\vec{u}'$ into $\vec{u}'_1, \ldots, \vec{u}'_q$ is bounded by $n^{O(q^2c)}$.

For all the other jobs in $\mathcal{J}_{i+2} \cap J(a_{i,t})$ that have $p_j < \ell_{i+3}$, because they are not position-crossing, we can assume their position (in the final solution) has intersection with only one interval of I_{i+2}. We also use the assumption that there is a near optimum solution in which they are not scheduled in their head or tail. So for the jobs in $\mathcal{J}_{i+2} \cap J(a_{i,t})$ with processing time less than ℓ_{i+3} we can re-define their span to a guessed sub-interval of $a_{i+1,t'+\gamma}$ at level I_{i+2}; these guesses define the qc-dimensional vectors $\vec{w}_\gamma$ for each interval $a_{i+1,t'+\gamma}$ (how many loose jobs from $\mathcal{J}_{i+2} \cap J(a_{i,t})$ with $p_j < \ell_{i+3}$ have their span redefined to be one of the q sub-intervals of $a_{i+1,t'+\gamma}$ at level $i+2$). Observe that, by only knowing how many of w_σ many jobs with processing times p_σ are scheduled in each interval $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ in the optimum solution, we would be able to detect which job is in which interval. The reason is that we know for each $p_\sigma \in P$, all jobs with processing time p_σ are scheduled based on earliest deadline first rule, which basically says that at any time when there are two jobs with the same processing time available the one with earliest deadline would be scheduled first.

Note that the jobs in $J(a_{i,t}) \cap \mathcal{J}_{\geq (i+3)}$ all have processing time at most

ℓ_{i+3} and their spans are completely inside one of intervals $a_{i+1,t'+1}$, ..., $a_{i+1,t'+q}$. These jobs will be passed down to the corresponding smaller sub-problems. So for each given $\vec{v}$ and $\vec{u}$, we consider all guesses S_0, S_1 and consider the resulting $\vec{u}', \vec{v}'$ and any possible way of breaking $\vec{u}'$, and $\vec{w}$ into q parts, we check:

$$A[a_{i+1,t'+1}, \vec{v}'_1, \vec{u}'_1+\vec{w}_1]+A[a_{i+1,t'+2}, \vec{v}'_2, \vec{u}'_2+\vec{w}_2]+\ldots+A[a_{i+1,t'+q}, \vec{v}'_q, \vec{u}'_q+\vec{w}_q]+s_0+s_1,$$

where s_0, s_1 are the sizes of the subsets S_0, S_1 of jobs with processing time $p_j \geq \ell_{i+3}$ guessed from $J(a_{i,t}) \cap \mathfrak{J}_{i+2}$ and those from $\vec{u}$ with processing time $p_j \geq \ell_{i+3}$. We would choose the maximum over all guesses $S_0 \subseteq \mathfrak{J}_{i+2} \cap J(a_{i,t})$, S_1, and all possible ways to distribute jobs with $p_j < \ell_{i+3}$ to create $\vec{u}'_\gamma$ and $\vec{w}'_\gamma$ as described above.

Note that to fill each entry $A[a_{i,t}, \vec{v}, \vec{u}]$ the number of jobs from $\mathfrak{J}_{i+2} \cap J(a_{i,t})$ plus jobs from $\vec{u}$ with processing time bigger than ℓ_{i+3} possible to be processed in $a_{i,t}$ would be at most q^3, because of their lengths. So we could have at most $n^{O(q^3 c)}$ many different $\vec{v}'$ to consider. For $\vec{u}'$ and $\vec{w}$ we would have at most $n^{O(q^2 c)}$ many ways to distribute each of them into q many qc-dimensional vectors. This means that we can fill the whole table in time at most $\Gamma k q n^{O(q^3 c)} = n^{O(\varepsilon^{-6} c)} \log_q T$, where Γ is the running time of the **PTAS** for Theorem 9, which is at most $2^{\varepsilon^{-1} \log^{-4}(1/\varepsilon)} + \text{Poly}(n)$. So the total time will be $n^{O(\varepsilon^{-6} c)} \log T$. □

3.3.4 Extension to $m = O(1)$ Machines

We show how to extend the result of Theorem 3 to $m = O(1)$ machines. We first do the randomized hierarchical decomposition of time line $[0, T]$ and define the classes of jobs $\mathfrak{J}_0, \mathfrak{J}_1, \ldots$ as before. Lemmas 6 and 7 can be adjusted to show that there is a solution with no span-crossing or position-crossing jobs of value at least $(1 - O(\varepsilon))$opt. Lemma 8 still holds for each machine. So we only need to explain how to change the DP for Theorem 11. Our dynamic program will be similar, except that for each interval $a_{i,t}$ sub-problems are defined based on m vectors $\vec{v}^1, \vec{v}^2, \ldots, \vec{v}^m$ corresponding to the blocked areas of the interval over machines $1, \ldots, m$ as well as vector $\vec{u}$. The sub-problems are stored in entries $A[a_{i,t}, \vec{v}^1, \vec{v}^2, \ldots, \vec{v}^m, \vec{u}]$ where each $\vec{v}^{i'}$ is a q^2-dimensional vector describing the blocked areas of $a_{i,t}$ on machine i' using jobs from $\mathcal{J}_{\leq i+2}$. Vector $\vec{u}$ as before is a (qc)-dimensional vector describing (for each $1 \leq \sigma \leq c$) the number of jobs of size p_σ that their span is redefined to one of the q sub-intervals that $a_{i,t}$ will be divided into, on any of the machines. So the number of sub-problems will be $(kn)n^{(m+c)q^2}$. At each step of the recursion, to fill in the entry $A[a_{i,t}, \vec{v}^1, \vec{v}^2, \ldots, \vec{v}^m, \vec{u}]$ we have to make similar guesses as before, except that now we have to decide on which of the m machines we schedule them. For the sets S_0, S_1 guessed from tight jobs and loose jobs from $\mathcal{J}_{i+2} \cap J(a_{i,t})$, we have $|\Psi|^{2q^3}$ guesses and for each of guesses another m options to decide the machines. So we will have $n^{O(mq^3c)}$ guesses. The number of guesses to break $\vec{u}$ to $\vec{u}'_1, \ldots, \vec{u}'_q$ will be the same. The rest of the computation of the entry is independent of the machines as we don't schedule any more jobs at this point. Hence, the total complexity of computing the entries of the DP table will be $O(\Gamma\varepsilon^{-2}kn^{O(mcq^3)}) = \varepsilon^{-3}n^{O(mc\varepsilon^{-6})}\log T$ (again noting that Γ being the running time of algorithm of Theorem 9) and we obtain a $(1 - O(c\varepsilon))$-approximation. For fixed m and c and for a given $\varepsilon' > 0$ one can choose $\varepsilon = \varepsilon'/c$ to obtain a $(1 - \varepsilon')$-approximation in time $n^{O(mc^7\varepsilon'^{-6})}\log T$.

If all p_j's are bounded polynomially in n then we can also use Theorem 3 to obtain a bi-criteria $(1 - \epsilon, 1 + \epsilon)$ quasi-polynomial time approximation. For simplicity consider the case of a single machine ($m = 1$). Given $\varepsilon' > 0$, we scale the processing times up to the nearest power of $(1 + \varepsilon')$. So we will have $c = O(\log n/\varepsilon')$ many distinct processing times. We the run the algorithm of Theorem 3 with $\varepsilon = \frac{\varepsilon'}{c} = \frac{\varepsilon'^2}{\log n}$. This will give a $(1 + O(\varepsilon c))$-approximation which we can run on a machine with $(1 + \varepsilon')$-speedup to compensate for the scaled-up processing times (so each scaled job will still finish by its deadline on the faster machine). Since $\varepsilon c = \frac{\varepsilon'^2}{\log n} \cdot \frac{\log n}{\varepsilon'} = \varepsilon'$, we obtain a $(1 - \varepsilon')$-approximation on $(1+\varepsilon')$-speedup machine in time $n^{O(\varepsilon'^{-13}\log^7 n)}$ (as mentioned earlier a stronger form of this, i.e. for weighted setting was already known [50]).

3.4 Cutting heads and tails: Proof of Lemma 9

We focus on optimum solution $O = \text{OPT}'$ and show how to modify O so that none of the jobs in the modified instance are scheduled in their head part without much loss in the throughput. For simplicity, we assume that J only contains the set of jobs scheduled in O. We basically want to construct another solution O'' by changing O such that in O'' the position of each loose job with processing time p has no intersection with its "head" part and at the same time its total profit is still comparable to O, which allows us to remove "head" part and still have a feasible solution with the desired total profit.

For each job j, recall that $span_j = [r_j, d_j]$, and if $j \in \mathcal{J}_i$ and $span_j$ has intersection with $a_{i,t_j}, ..., a_{i,t'_j}$ from I_i then $head_j = span_j \cap a_{i,t_j}$ and $tail_j = span_j \cap a_{i,t'_j}$. We let $\overline{span}_j = span_j - (head_j \cup tail_j)$ be the reduced span of j. Our goal is to modify O so that every loose job j is scheduled in O in $\overline{span}_j$. The idea of the proof is to move each loose job j with processing time p scheduled in its head (or tail) to be re-scheduled in $\overline{span}_j$ if there is empty space for it there. If not, and if we can remove some larger (w.r.t. processing time) jobs in $\overline{span}_j$ to make room for j and possibly other loose jobs whose head is in a_{i,t_j} we do so. Otherwise, it means that the entire λ intervals starting from a_{i,t_j} which $span_j$ has intersection with is relatively packed with jobs of size p or smaller. We want to argue that in this case even if we remove j (and all other loose jobs in a_{i,t_j}) we can "charge" them to the collection of many jobs scheduled in the next λ intervals; hence the loss will be relatively small. However, we cannot do this simple charging argument since the intervals to which we charge (for the jobs removed) are not all disjoint; hence a job that remains might be charged multiple times (due to the hierarchy of the intervals we have defined). Nevertheless, we show a careful charging scheme that will ensure the total loss for jobs, that cannot be rescheduled in their reduced span, is still relatively small.

Proof. Consider $O = \text{OPT}'$ and assume that J is simply the set of jobs in O. We focus on the loose jobs of size p that their position in O has intersection with their "head" (argument is similar for the case of "tail" we just do the reverse order). We traverse all the loose jobs of size p in J in the order of their position in O from the latest to the earliest. For each such job $j \in J$ assume $j \in \mathfrak{J}_i$ for some $0 \leq i \leq k$ and $span_j$ has intersection with $a_{i,t_j}, ..., a_{i,t'_j}$ from I_i. Note that since $j \in \mathfrak{J}_i$ it means $t'_j - t_j \geq \lambda$. While traversing j if its position in O has intersection with $head_j$ we add it to set X_{i,t_j} (which is initially empty) corresponding to interval $a_{i,t_j} \in I_i$ and try to move it to $\overline{span}_j$ if possible (without changing the position of any other job). This means if there is empty space in $\overline{span_j}$ we try to re-schedule j there. If this is not possible, then temporarily remove it from O (to make room for the rest of the jobs currently running in their head) and add it to set X'_{i,t_j} (which is initially empty too).

After changing the position of some loose jobs and removing some others, it is obvious that the position of each scheduled loose job of size p has no intersection with its head in the current solution which we denote by O'. Observe that for each interval $a_{i,t} \in I_i$ for $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$, we have $X'_{i,t} \subseteq X_{i,t}$ and $|X'_{i,t}| = x'_{i,t} \leq |X_{i,t}| = x_{i,t}$. Also if $x'_{i,t} > 0$, then there is no empty space for a job with processing time p in the following $\lambda - 1$ intervals of I_i, i.e. if we define $Y_{i,t} = a_{i,t+1} \cup ... \cup a_{i,t+\lambda-1}$, there is no empty space of size p in $Y_{i,t}$. This uses the fact that for any job like j whose head is $a_{i,t}$, its span contains all of $Y_{i,t}$. So $x'_{i,0} > 0$ means there are such jobs of size p (whose head is in $a_{i,t}$) and they could not be moved to any space in $Y_{i,t}$.

Consider interval $a_{i,t}$ for any $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$. We define $y_{i,t} = \frac{|Y_{i,t}|}{p} = \frac{(\lambda-1)\cdot\ell_i}{p}$, and $A_{i,t}$ as the set consisting of all $a_{i',t'}$ such that $Y_{i,t} \cap Y_{i',t'} \neq \emptyset$ and

- $i' > i$, or
- $i' = i$ and $t' > t$.

So those in $A_{i,t}$ are the intervals $a_{i',t'}$ whose Y set has overlap with that of $a_{i,t}$ and either $a_{i',t'}$ is at a finer level of hierarchy, or is at the same level i but at a later time. We then partition $A_{i,t}$ into two sets $A^1_{i,t}$ and $A^2_{i,t}$:

- if $a_{i',t'} \subseteq a_{i,t}$ then $a_{i',t'} \in A^1_{i,t}$,

- else $a_{i',t'} \in A^2_{i,t}$.

Observe that for each $a_{i',t'} \in A^2_{i,t}$ we have $a_{i',t'} \subseteq Y_{i,t}$ and this means that removing any job from $a_{i',t'} \in A^2_{i,t}$ would make an empty room for a job in $X'_{i,t}$.

Next lemma would provide an important fact about intervals whose Y parts are not disjoint and basically provides an upper bound on the number of jobs removed temporarily from all intervals in $A_{i,t}$ during the first phase while converting O to O':

Lemma 10. *For each $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$ with $x'_{i,t} > 0$:*

- $x'_{i,t} + \sum_{a_{i',t'} \in A^1_{i,t}} x'_{i',t'} \leq \frac{3}{\lambda} \cdot y_{i,t}$,
- $x'_{i,t} + \sum_{a_{i',t'} \in A^2_{i,t}} x'_{i',t'} \leq \frac{3}{\lambda} \cdot y_{i,t}$.

We defer the proof of this lemma to later.

Corollary 2. *For each $0 \leq i \leq k$, and $0 \leq t \leq \frac{T}{\ell_i}$ with $x'_{i,t} > 0$:*

$$x'_{i,t} + \sum_{a_{i',t'} \in A_{i,t}} x'_{i',t'} \leq \frac{6}{\lambda} \cdot y_{i,t}$$

Next we traverse all intervals on a specific order and change O' to O'' so that we can compare its total profit with OPT' while still no scheduled job has intersection with its "head" part. For each i from 0 to k and for each t from 0 to $\frac{T}{\ell_i}$, if $x'_{i,t} > 0$ do the following:

If (and while) the processing time of the biggest job which is currently scheduled in $Y_{i,t}$ is more than p, and $X'_{i,t}$ is not empty yet, remove that biggest job from O', add it to set $R_{i,t}$ (which is initially empty) and add as many jobs from $X'_{i,t}$ to O' as possible in the empty space which is just freed up by removing that big job. We repeat this as long as $X'_{i,t} \neq \emptyset$ and the size of the biggest job currently scheduled in $Y_{i,t}$ is larger than p. Note that jobs in $X'_{i,t}$ all have processing time p and able to be scheduled in whole $Y_{i,t}$ since their span contains $Y_{i,t}$. At the end, if $X'_{i,t} \neq \emptyset$ and the processing time of the biggest remaining job in $Y_{i,t}$ is no more than p (or in the case it was initially at most p), add all the remaining jobs in $X'_{i,t}$ to $R_{i,t}$, and define $p'_{i,t}$ as the processing time of the smallest job in $R_{i,t}$ and set $\alpha_{i,t} = \lfloor \frac{p'_{i,t}}{p} \rfloor$. Note that all the jobs remaining in $Y_{i,t}$ would have processing time at most $p'_{i,t}$.

Now we have our solution O'' which we claim has near optimum total profit. First observe that no loose job of size p in O'' is scheduled having intersection with its head. Also, no job is moved to its head. Note that for all $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$, $R_{i,t}$ would contain all the jobs which are actually removed from optimum solution O:

$$O = O'' \cup \bigcup_{i,t} R_{i,t}$$

Let's denote by $S_{i,t}$ the set of jobs scheduled inside $Y_{i,t}$ in solution O'' for each $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$. Then the union of all these sets for all intervals would be a subset of O'':

$$\bigcup_{i,t} S_{i,t} \subseteq O'' \quad \Rightarrow \quad |\bigcup_{i,t} S_{i,t}| \leq |O''|$$

Our goal is to show that $|\bigcup_{i,t} R_{i,t}| \leq \frac{60}{\lambda} |\bigcup_{i,t} S_{i,t}|$ which completes the proof of Lemma 9:

$$|O''| = |O| - |\bigcup_{i,t} R_{i,t}| \geq |O| - \frac{60}{\lambda} \cdot |\bigcup_{i,t} S_{i,t}| \geq |O| - \frac{60}{\lambda} \cdot |O''| \geq (1 - \frac{60}{\lambda})|O|$$

The next lemma which upper bounds $|R_{i,t}|$ by a small fraction of $|S_{i,t}|$ can be proved using the "simple" charging scheme explained at the beginning of this section. We defer the proof of this lemma to later.

Lemma 11. *For each $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$ with $x'_{i,t} > 0$:*

$$|R_{i,t}| \leq \frac{30}{\lambda} |S_{i,t}|$$

This means that the number of jobs removed from O for each interval $a_{i,t}$ (namely $|R_{i,t}|$), is at most $\frac{30}{\lambda}$ of the number of jobs scheduled in interval $Y_{i,t}$ (namely $|S_{i,t}|$). If it was the case that for any two intervals a_{i_1,t_1} and a_{i_2,t_2}, we have $S_{i_1,t_1} \cap S_{i_2,t_2} = \emptyset$, then Lemma 11 would be enough to complete the proof of Lemma 9. But the problem is that for any two different intervals a_{i_1,t_1} and a_{i_2,t_2}, by definition, R_{i_1,t_1} and R_{i_2,t_2} are disjoint but S_{i_1,t_1} and S_{i_2,t_2} could have intersection. In other words we might have some intervals a_{i_1,t_1}, a_{i_2,t_2} with $Y_{i_1,t_1} \cap Y_{i_2,t_2} \neq \emptyset$ which means $S_{i_1,t_1} \cap S_{i_2,t_2} \neq \emptyset$. The next lemma will help us to "uncross" those Y's:

Lemma 12. *For each interval $a_{i,t}$ with $x'_{i,t} > 0$, we can partition $A_{i,t}$ into two parts A_1 and A_2 such that*

$$|R_{i,t} \cup \bigcup_{a_{i',t'} \in A_1} R_{i',t'}| \leq \frac{60}{\lambda} |S_{i,t} \setminus \bigcup_{a_{i',t'} \in A_2} S_{i',t'}|$$

Using Lemma 12 we can partition all intervals into a number of disjoint groups such that for each group the number of total jobs removed from O is a $\frac{60}{\lambda}$ fraction of the number of jobs scheduled in O'' in that group.

Suppose $a_{i,t}$ is an interval with the lowest i value (breaking the ties with equal i by taking the smallest t) with $x'_{i,t} > 0$. Using Lemma 12 we find some $A_1 \subseteq A_{i,t}$ and the first group G_1 of intervals we define will be $G_1 = \{a_{i,t}\} \cup A_1$. If we denote $R(G_1) = R_{i,t} \cup \bigcup_{a_{i',t'} \in A_1} R_{i',t'}$ and $S(G_1) = S_{i,t} \setminus \bigcup_{a_{i',t'} \in A_2} S_{i',t'}$ then using Lemma 12: $|R(G_1)| \leq \frac{60}{\lambda}|S(G_1)|$. Also $S(G_1) \cap S_{i',t'} = \emptyset$ for any $a_{i',t'} \notin A_1 \cup \{a_{i,t}\}$ for the following reason: if $a_{i',t'} \in A_2$ then clearly $S(G_1) \cap S_{i',t'} = \emptyset$ from definition of $S(G_1)$; if $a_{i',t'} \notin A_1 \cup A_2$ then $Y_{i',t'}$ has no intersection with $Y_{i,t}$ and hence $S(G_1) \cap S_{i',t'} = \emptyset$. Note that if $A_{i,t} = \emptyset$, then we can use Lemma 11, we have $G_1 = \{a_{i,t}\}$ and $|R(G_1)| \leq \frac{60}{\lambda}|S(G_1)|$, holds for this case too.

So we can remove group G_1 along with the corresponding sets $R(G_1)$ and $S(G_1)$ and continue doing the same for the remaining intervals to construct the next group. Observe that at each step by removing a group of intervals, the remaining intervals are not changed and this allows us to be able to do the same process for them. Finally we obtain a collection of groups $G_1, G_2, \ldots$ where for each G_i: $|R(G_i)| \leq \frac{60}{\lambda}|S(G_i)|$ and the sets $S(G_i)$'s are disjoint. Since $\bigcup_i S(G_i)$ is a subset of all jobs scheduled in O'' and $\bigcup_i R(G_i)$ is the set of all jobs removed from O to obtain O'', the proof of Lemma 9 follows.

□

3.4.1 Proof of Lemma 10

Proof. Recall that the first step of converting O to O' was to traverse all the scheduled jobs based on their position in O. To prove the first statement of Lemma 10, note that all the jobs removed while traversing $a_{i,t}$ and $A^1_{i,t}$, have processing time p and had initially intersection with interval $a_{i,t}$ with length ℓ_i in O. Observe that their length is p and so all could be scheduled in an interval with length $\ell_i + p$. Assuming $\lambda > 3$ we have:

$$x'_{i,t} + \sum_{a_{i',t'} \in A^1_{i,t}} x_{i',t'} \leq \frac{\ell_i + p}{p} \leq \frac{2\ell_i}{p} \leq \frac{2\ell_i}{p} \cdot \frac{3(\lambda - 1)}{2\lambda} = \frac{3}{\lambda} \cdot y_{i,t}. \tag{3.1}$$

To prove the second statement, observe that while traversing the jobs in $A^2_{i,t}$ we have temporarily removed $\sum_{a_{i',t'} \in A^2_{i,t}} x'_{i',t'}$ many jobs with processing time p and they make room for the same number of jobs (of size p) in $a_{i,t}$. Note that all $x'_{i,t}$ many jobs which are temporarily removed while traversing $a_{i,t}$ could be scheduled in the whole interval $Y_{i,t}$ (as their span contains $Y_{i,t}$). So from at most $\frac{\ell_i+p}{p}$ many jobs initially intersecting with interval $a_{i,t}$, at most $\frac{\ell_i+p}{p} - \sum_{a_{i',t'} \in A^2_{i,t}} x_{i',t'}$ many of them would be temporarily removed while traversing $a_{i,t}$:

$$x'_{i,t} + \sum_{a_{i',t'} \in A^2_{i,t}} x_{i',t'} \leq \frac{\ell_i + p}{p} \leq \frac{3}{\lambda} \cdot y_{i,t}. \tag{3.2}$$

We only need to sum up inequalities (3.1) and (3.2) to prove Corollary 2:

$$x'_{i,t} + \sum_{a_{i',t'} \in A_{i,t}} x'_{i',t'} \leq \big(x'_{i,t} + \sum_{a_{i',t'} \in A^1_{i,t}} x_{i',t'}\big) + \big(x'_{i,t} + \sum_{a_{i',t'} \in A^2_{i,t}} x_{i',t'}\big) \leq \frac{6}{\lambda} \cdot y_{i,t}$$

□

3.4.2 Proof of Lemma 11

Proof. Fix some $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$. First observe that for each interval $a_{i,t}$ with positive $x'_{i,t}$, we have removed at most $\lceil \frac{x'_{i,t}}{\alpha_{i,t}} \rceil$ many jobs from O to obtain O''. This is obvious if $\alpha = 1$. For $\alpha > 1$ note that for each job bigger than p removed from $Y_{i,t}$ we could schedule at least $\alpha_{i,t}$ many jobs of size p.

$$|R_{i,t}| \leq \lceil \frac{x'_{i,t}}{\alpha_{i,t}} \rceil \leq \frac{2x'_{i,t}}{\alpha_{i,t}} \tag{3.3}$$

Also note that the length of $Y_{i,t}$ is $(\lambda - 1) \cdot \ell_i$ and all the jobs inside $Y_{i,t}$ in solution O'' have processing time at most $p'_{i,t} \leq 2p\alpha_{i,t}$ and between any two consecutive scheduled job there can be at most $p'_{i,t}$ empty space. So the time between the starting time of each two consecutive scheduled job in $Y_{i,t}$ could not be more than $2p'_{i,t}$.

$$|S_{i,t}| \geq \lfloor \frac{(\lambda-1)\ell_i}{4p\alpha_{i,t}} \rfloor \geq \frac{y_{i,t}}{5\alpha_{i,t}} \tag{3.4}$$

Considering Lemma 10 we have:

$$x'_{i,t} \leq \frac{3}{\lambda} \cdot y_{i,t} \tag{3.5}$$

To complete the proof of Lemma 11 we only need to combine Inequalities (3.3), (3.4), and (3.5):

$$|R_{i,t}| \leq \frac{2x'_{i,t}}{\alpha_{i,t}} \leq \frac{2}{\alpha_{i,t}} \cdot \frac{3y_{i,t}}{\lambda} = \frac{30}{\lambda} \cdot \frac{y_{i,t}}{5\alpha_{i,t}} \leq \frac{30}{\lambda}|S_{i,t}|$$

□

3.4.3 Proof of Lemma 12

Proof. Fix some $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$ and suppose we have sorted all intervals $a_{i',t'} \in A_{i,t}$ based on their $\alpha_{i',t'}$ values (in descending order) and for simplicity rename them so that $A_{i,t} = \{a_{i_1,t_1}, a_{i_2,t_2}, \ldots, a_{i_r,t_r}\}$ where $\alpha_{i_1,t_1} \geq \alpha_{i_2,t_2} \geq \ldots \geq \alpha_{i_r,t_r}$.

Suppose h is the highest index where $\alpha_{i_h,t_h} \geq \alpha_{i,t}$ ($h = 0$ if there is no such index). We claim that there is an index s, $h \leq s \leq r$, such that the statement of Lemma 12 holds for $A_1 = \{a_{i_1,t_1}, \ldots, a_{i_s,t_s}\}$ and $A_2 = \{a_{i_{s+1},t_{s+1}}, \ldots, a_{i_r,t_r}\}$. By way of contradiction suppose that the statement of Lemma 12 is not valid for any s, $h \leq s \leq r$. Thus:

$$|R_{i,t} \cup \bigcup_{u=1}^{s} R_{i_u,t_u}| > \frac{60}{\lambda}|S_{i,t} \setminus \bigcup_{u=s+1}^{r} S_{i_u,t_u}| \tag{3.6}$$

Also based on Lemma 10 we have:

$$x'_{i,t} + \sum_{u=1}^{r} x'_{i_u,t_u} \leq \frac{6}{\lambda} \cdot y_{i,t} = \frac{6}{\lambda} \cdot \frac{|Y_{i,t}|}{p} \tag{3.7}$$

We are going to show that we cannot have Inequalities (3.7) and (3.6) for all s, $h \leq s \leq r$ at the same time and reach a contradiction. First of all to find an upper bound for the left side of Inequality (3.6), observe that, by definition, for any two intervals $a_{i,t}$ and $a_{i',t'}$ there is no intersection between $R_{i,t}$ and $R_{i',t'}$. By using Inequality (3.3), for each s, $h \leq s \leq r$ we have:

$$|R_{i,t} \cup \bigcup_{u=1}^{s} R_{i_u,t_u}| = |R_{i,t}| + \sum_{u=1}^{s} |R_{i_u,t_u}| \leq \frac{2x'_{i,t}}{\alpha_{i,t}} + \sum_{u=1}^{s} \frac{2x'_{i_u,t_u}}{\alpha_{i_u,t_u}} \tag{3.8}$$

To have a lower bound for the right side of Inequality (3.6) we are going to define $Y^*_{i_u,t_u}$ for each u, $h < u \leq r$ and $Y^*_{i,t}$:

$$Y^*_{i_r,t_r} = Y_{i_r,t_r} \cap Y_{i,t}$$

$$h < u < r \quad \Rightarrow \quad Y^*_{i_u,t_u} = \big(Y_{i_u,t_u} \cap Y_{i,t}\big) \setminus \big(Y^*_{i_{u+1},t_{u+1}} \cup \ldots \cup Y^*_{i_r,t_r}\big)$$

$$Y^*_{i,t} = Y_{i,t} \setminus \big(Y^*_{i_{h+1},t_{h+1}} \cup \ldots \cup Y^*_{i_r,t_r}\big)$$

Note that $Y^*_{i,t}$ along with all $Y^*_{i_u,t_u}$'s are a partition of $Y_{i,t}$:

$$|Y_{i,t}| = |Y^*_{i,t}| + \sum_{u=h+1}^{r} |Y^*_{i_u,t_u}| \tag{3.9}$$

Also note that for each u, $h < u \leq r$ jobs scheduled inside $Y^*_{i_u,t_u}$ in O'' have processing time at most $p'_{i_u,t_u} \leq 2p\alpha_{i_u,t_u}$ and the empty space between any two consecutive scheduled job is no more than p'_{i_u,t_u} too (otherwise we were able to add some more jobs from $X^p_{i,t}$ to O''), and jobs scheduled inside $Y^*_{i,t}$ have processing time at most $p'_{i,t} \leq 2p\alpha_{i,t}$. So for each s, $h \leq s \leq r$ we have:

$$|S_{i,t} \setminus \bigcup_{u=s+1}^{r} S_{i_u,t_u}| \geq \lfloor \frac{|Y^*_{i,t}|}{4p\alpha_{i,t}} \rfloor + \sum_{u=h+1}^{s} \lfloor \frac{|Y^*_{i_u,t_u}|}{4p\alpha_{i_u,t_u}} \rfloor \geq \frac{|Y^*_{i,t}|}{5p\alpha_{i,t}} + \sum_{u=h+1}^{s} \frac{|Y^*_{i_u,t_u}|}{5p\alpha_{i_u,t_u}} \tag{3.10}$$

The only thing we need to prove to complete the proof of the Lemma 12 is that there is an index s, $h \leq s \leq r$ such that:

$$\frac{x'_{i,t}}{\alpha_{i,t}} + \sum_{u=1}^{s} \frac{x'_{i_u,t_u}}{\alpha_{i_u,t_u}} \leq \frac{6}{\lambda} \left(\frac{|Y^*_{i,t}|}{p\alpha_{i,t}} + \sum_{u=h+1}^{s} \frac{|Y^*_{i_u,t_u}|}{p\alpha_{i_u,t_u}} \right) \tag{3.11}$$

Combining Inequalities (3.8), (3.10), and (3.11) completes the proof:

$$|R_{i,t} \cup \bigcup_{u=1}^{s} R_{i_u,t_u}| \leq \frac{2x'_{i,t}}{\alpha_{i,t}} + \sum_{u=1}^{s} \frac{2x'_{i_u,t_u}}{\alpha_{i_u,t_u}}$$

$$\leq \frac{12}{\lambda}\Big(\frac{|Y^*_{i,t}|}{p\alpha_{i,t}} + \sum_{u=h+1}^{s} \frac{|Y^*_{i_u,t_u}|}{p\alpha_{i_u,t_u}}\Big) \leq \frac{60}{\lambda}|S_{i,t} \setminus \bigcup_{u=s+1}^{r} S_{i_u,t_u}|$$

Thus, we now prove Inequality (3.11). We consider two cases. For the first case suppose that $h = r$, which means that $\alpha_{i,t} \leq \alpha_{i_u,t_u}$ for all $1 \leq u \leq r$. Note that in this case $Y^*_{i,t} = Y_{i,t}$ and Inequality (3.11) would be proved using the inequality (3.7):

$$\frac{x'_{i,t}}{\alpha_{i,t}} + \sum_{u=1}^{s} \frac{x'_{i_u,t_u}}{\alpha_{i_u,t_u}} \leq \frac{1}{\alpha_{i,t}}\big(x'_{i,t} + \sum_{u=1}^{s} x'_{i_u,t_u}\big) \leq \frac{6}{\lambda}\frac{|Y^*_{i,t}|}{p\alpha_{i,t}} \tag{3.12}$$

Hence we suppose $h < r$ and for the sake of contradiction suppose that Inequality (3.11) is not true for any value of s. So for all s, $h \leq s \leq r$ we have:

$$\frac{x'_{i,t}}{\alpha_{i,t}} + \sum_{u=1}^{s} \frac{x'_{i_u,t_u}}{\alpha_{i_u,t_u}} > \frac{6}{\lambda}\Big(\frac{|Y^*_{i,t}|}{p\alpha_{i,t}} + \sum_{u=h+1}^{s} \frac{|Y^*_{i_u,t_u}|}{p\alpha_{i_u,t_u}}\Big) \tag{3.13}$$

What we do is, for each value of s, $h \leq s \leq r$, we multiply both sides of Inequality (3.13) and sum all of them to derive a contradiction. For $s = h$, multiply both sides of Inequality (3.13) by $\alpha_{i,t} - \alpha_{i_{h+1},t_{h+1}}$ and for $s = r$ multiply both sides by α_{i_r,t_r}, and for every other s, $h < s < r$ multiply both sides of Inequality (3.13) associated with s by $\alpha_{i_s,t_s} - \alpha_{i_{s+1},t_{s+1}}$. Note that considering the definition of h and the fact that $h < r$, we have $\alpha_{i,t} > \alpha_{i_{h+1},t_{h+1}} \geq \ldots \geq \alpha_{i_r,t_r} \geq 1$, so all the coefficients are non-negative (and in fact the first one is positive):

$$(\alpha_{i,t} - \alpha_{i_{h+1},t_{h+1}}) \quad \times \quad \left(\frac{x'_{i,t}}{\alpha_{i,t}} + \frac{x'_{i_1,t_1}}{\alpha_{i_1,t_1}} + \frac{x'_{i_2,t_2}}{\alpha_{i_2,t_2}} + \ldots + \frac{x'_{i_h,t_h}}{\alpha_{i_h,t_h}} > \frac{6}{\lambda}\Big(\frac{|Y^*_{i,t}|}{p\alpha_{i,t}}\Big)\right)$$

$$(\alpha_{i_{h+1},t_{h+1}} - \alpha_{i_{h+2},t_{h+2}}) \quad \times \quad \left(\frac{x'_{i,t}}{\alpha_{i,t}} + \frac{x'_{i_1,t_1}}{\alpha_{i_1,t_1}} + \ldots + \frac{x'_{i_{h+1},t_{h+1}}}{\alpha_{i_{h+1},t_{h+1}}} > \frac{6}{\lambda}\Big(\frac{|Y^*_{i,t}|}{p\alpha_{i,t}} + \frac{|Y^*_{i_{h+1},t_{h+1}}|}{p\alpha_{i_{h+1},t_{h+1}}}\Big)\right)$$

$$(\alpha_{i_{h+2},t_{h+2}} - \alpha_{i_{h+3},t_{h+3}}) \quad \times \quad \left(\frac{x'_{i,t}}{\alpha_{i,t}} + \frac{x'_{i_1,t_1}}{\alpha_{i_1,t_1}} + \ldots + \frac{x'_{i_{h+2},t_{h+2}}}{\alpha_{i_{h+2},t_{h+2}}} > \frac{6}{\lambda}\Big(\frac{|Y^*_{i,t}|}{p\alpha_{i,t}} + \frac{|Y^*_{i_{h+1},t_{h+1}}|}{p\alpha_{i_{h+1},t_{h+1}}} + \frac{|Y^*_{i_{h+2},t_{h+2}}|}{p\alpha_{i_{h+2},t_{h+2}}}\Big)\right)$$

$$\vdots$$

$$(\alpha_{i_s,t_s} - \alpha_{i_{s+1},t_{s+1}}) \quad \times \quad \left(\frac{x'_{i,t}}{\alpha_{i,t}} + \frac{x'_{i_1,t_1}}{\alpha_{i_1,t_1}} + \ldots + \frac{x'_{i_s,t_s}}{\alpha_{i_s,t_s}} > \frac{6}{\lambda}\Big(\frac{|Y^*_{i,t}|}{p\alpha_{i,t}} + \frac{|Y^*_{i_{h+1},t_{h+1}}|}{p\alpha_{i_{h+1},t_{h+1}}} + \ldots + \frac{|Y^*_{i_s,t_s}|}{p\alpha_{i_s,t_s}}\Big)\right)$$

$$\vdots$$

$$(\alpha_{i_{r-1},t_{r-1}}-\alpha_{i_r,t_r}) \quad \times \quad \left(\frac{x'_{i,t}}{\alpha_{i,t}}+\frac{x'_{i_1,t_1}}{\alpha_{i_1,t_1}}+\ldots+\frac{x'_{i_{r-1},t_{r-1}}}{\alpha_{i_{r-1},t_{r-1}}} > \frac{6}{\lambda}\left(\frac{|Y^*_{i,t}|}{p\alpha_{i,t}}+\frac{|Y^*_{i_{h+1},t_{h+1}}|}{p\alpha_{i_{h+1},t_{h+1}}}+\ldots+\frac{|Y^*_{i_{r-1},t_{r-1}}|}{p\alpha_{i_{r-1},t_{r-1}}}\right)\right)$$

$$(\alpha_{i_r,t_r}) \quad \times \quad \left(\frac{x'_{i,t}}{\alpha_{i,t}}+\frac{x'_{i_1,t_1}}{\alpha_{i_1,t_1}}+\ldots+\frac{x'_{i_r,t_r}}{\alpha_{i_r,t_r}} > \frac{6}{\lambda}\left(\frac{|Y^*_{i,t}|}{p\alpha_{i,t}}+\frac{|Y^*_{i_{h+1},t_{h+1}}|}{p\alpha_{i_{h+1},t_{h+1}}}+\ldots+\frac{|Y^*_{i_r,t_r}|}{p\alpha_{i_r,t_r}}\right)\right)$$

Now we sum up all these inequalities (with the corresponding coefficients) to reach a contradiction. Since all coefficients are ≥ 0 and the very first one is positive ($\alpha_{i,t} - \alpha_{i_{h+1},t_{h+1}} > 0$) this ensures that we have non-zero sum. Note that for each $1 \leq s \leq h$, term $\frac{x'_{i_s,t_s}}{\alpha_{i_s,t_s}}$ has appeared in the left hand side of all the above inequalities and so its coefficient in the sum would be the sum of all the coefficients:

$$(\alpha_{i,t}-\alpha_{i_{h+1},t_{h+1}})+(\alpha_{i_{h+1},t_{h+1}}-\alpha_{i_{h+2},t_{h+2}})+\ldots+(\alpha_{i_{r-1},t_{r-1}}-\alpha_{i_r,t_r})+(\alpha_{i_r,t_r}) = \alpha_{i,t}$$

This is the case for terms $\frac{x'_{i,t}}{\alpha_{i,t}}$ and $\frac{|Y^*_{i,t}|}{p\alpha_{i,t}}$ as well. Also for each s, $h < s \leq r$, terms $\frac{x'_{i_s,t_s}}{\alpha_{i_s,t_s}}$ and $\frac{|Y^*_{i_s,t_s}|}{p\alpha_{i_s,t_s}}$ have appeared in the left hand side and the right hand side of Inequality (3.13) associated with all values $s, s+1, s+2, \ldots, r$, respectively. So the coefficient for $\frac{x'_{i_s,t_s}}{\alpha_{i_s,t_s}}$ and $\frac{|Y^*_{i_s,t_s}|}{p\alpha_{i_s,t_s}}$ in the sum would be:

$$(\alpha_{i_s,t_s}-\alpha_{i_{s+1},t_{s+1}})+(\alpha_{i_{h+1},t_{h+1}}-\alpha_{i_{h+2},t_{h+2}})+\ldots+(\alpha_{i_{r-1},t_{r-1}}-\alpha_{i_r,t_r})+(\alpha_{i_r,t_r}) = \alpha_{i_s,t_s}$$

This means that the sum of all the inequalities written above can be simplified to:

$$\alpha_{i,t}\Big(\frac{x'_{i,t}}{\alpha_{i,t}}+\sum_{s=1}^{h}\frac{x'_{i_s,t_s}}{\alpha_{i_s,t_s}}\Big)+\sum_{s=h+1}^{r}\alpha_{i_s,t_s}\cdot\frac{x'_{i_s,t_s}}{\alpha_{i_s,t_s}} > \frac{6}{\lambda}\left(\alpha_{i,t}\frac{|Y^*_{i,t}|}{p\alpha_{i,t}}+\sum_{s=h+1}^{r}\alpha_{i_s,t_s}\frac{|Y^*_{i_s,t_s}|}{p\alpha_{i_s,t_s}}\right)$$

$$\Longrightarrow \alpha_{i,t}\Big(\frac{x'_{i,t}}{\alpha_{i,t}}+\sum_{s=1}^{h}\frac{x'_{i_s,t_s}}{\alpha_{i_s,t_s}}\Big)+\sum_{s=h+1}^{r}x'_{i_s,t_s} > \frac{6}{\lambda}\left(\frac{|Y^*_{i,t}|}{p}+\sum_{s=h+1}^{r}\frac{|Y^*_{i_s,t_s}|}{p}\right)$$

Considering that $\alpha_{i_1,t_1} \geq \alpha_{i_2,t_2} \geq \ldots \geq \alpha_{i_h,t_h} \geq \alpha_{i,t}$, and Equality (3.9) we have:

$$x'_{i,t}+\sum_{s=1}^{r}x'_{i_s,t_s} \geq \alpha_{i,t}\Big(\frac{x'_{i,t}}{\alpha_{i,t}}+\sum_{s=1}^{h}\frac{x'_{i_s,t_s}}{\alpha_{i_s,t_s}}\Big)+\sum_{s=h+1}^{r}x'_{i_s,t_s} > \frac{6}{\lambda}\cdot\frac{|Y^*_{i,t}|+\sum_{u=h+1}^{r}|Y^*_{i_u,t_u}|}{p} = \frac{6}{\lambda}\cdot\frac{|Y_{i,t}|}{p}$$

$$\Rightarrow \quad x'_{i,t}+\sum_{s=1}^{r}x'_{i_s,t_s} > \frac{6}{\lambda}\cdot\frac{|Y_{i,t}|}{p}$$

This contradicts Inequality (3.7), which was based on Lemma 10 for interval $a_{i,t}$. This contradiction show that for at least one value of s, Inequality (3.11) holds, which completes the proof of Lemma 12.

□

Chapter 4

Hierarchical Clustering

Recall that in Hierarchical Clustering the set of data points of input are represented as the vertices of a weighted graph $G = (V, E)$ where for any two nodes i and j, $w_{i,j}$ is the weight (similarity or dissimilarity) between the two data points. Then one can think of a hierarchical clustering as a tree T whose leaves are nodes of G and each internal node corresponds to the subset of nodes of the leaves in that subtree (hence root of T corresponds to V). For any two data points i and j we use $T_{i,j}$ to denote the subtree rooted at the least common ancestor (LCA) of i and j and $w_{i,j}$ represents the similarity between i, j.

Cohen-Addad et al. [31] considered Dasgupta's [32] objective function but for dissimilarity-based graphs, where $w_{i,j}$ is the weight of dissimilarity between two nodes i and j. In this version a good clustering should have larger $T_{i,j}$ when $w_{i,j}$ is relatively large. Here the objective is to maximize the following formula:

$$Rev(T) = \sum_{i,j \in [n]} \big(w_{i,j} \times |T_{i,j}| \big) \tag{4.1}$$

We build upon the work of [22] and present an algorithm that takes advantage of some conditions in which their algorithm fails to perform better. Since the final algorithm (and its analysis) is more complex, and for ease of exposition, we start with a simpler algorithm which achieves ratio 0.6929. Then through a series of improvements we show how we can get to 0.71604 ratio to prove Theorem 4 which is saying for hierarchical clustering on dissimilarity-based graphs, there is an approximation algorithm to maximize objective of (1.5) with ratio 0.71604.

We also introduce a new objective function for hierarchical clustering and present approximation algorithms for this new objective. More precisely, consider any tree T, we define $H_{i,j}$ as the number of common ancestors of i and j in T. In other words, when we think of the process of building the tree as a top-down procedure, $H_{i,j}$ is the step in which we decide to separate i and j from each other (they were decided to be together in $H_{i,j}-1$ many steps). Then for dissimilarity-based graphs, we propose to minimize the following objective:

$$Cost_H(T) = \sum_{i,j\in[n]} \big(w_{i,j} \times H_{i,j}\big). \tag{4.2}$$

The problem we are looking to solve here is to find a full binary tree with the minimum $Cost_H(.)$. One of the advantages of this objective function over the one introduced by Dasgupta [32] is when we have a complete graph with unit weights. All trees would have the same cost when we consider Dasgupta's objective function. But when we consider $Cost_H(T)$, balanced trees have the least cost.

It is easy to see that any algorithm that gives a balanced binary tree would have approximation ratio at most $O(\log n)$ since the height of such trees is $O(\log n)$. Furthermore, it is not hard to verify that the average-linkage algorithm would not perform well for this new objective function. The following example is an instance for which the cost of the solution of the average-linkage algorithm is at least $O(\frac{n}{\log n})$ times the cost of the optimum solution. Consider a graph with n vertices: $v_1, v_2, ..., v_n$. Then for each $2 \le j \le n$ and for each $1 \le i < j$ let $w_{i,j} = j-1$. In this graph the summation of all the edges would be $O(n^3)$ but running average linkage on this graph would result in a tree T with $Cost_H(T) = O(n^4)$ while the optimum tree (as well as any balanced binary tree) will have cost $O(n^3 \log n)$.

We are going to show that a top-down algorithm that chooses the approximated weighted max-cut at each step, would be a $\frac{4\alpha_{GW}}{4\alpha_{GW}-1}$-approximation algorithm to minimize $Cost_H(T)$, where α_{GW} is the ratio of the max-cut approximation algorithm. Considering that the best known approximation algorithm for weighted maximum cut problem has ratio $\alpha_{GW} = 0.8786$ [41], the ratio of this algorithm would be 1.3977. This will prove Theorem 5.

4.1 Prior Work

An alternative interpretation of Dasgupta's cost function is in terms of cuts. In a top-down approach at each step we must partition a set of items into two groups (recall that the optimal tree is binary). We can set a cost for each step such that the total cost would be the summation of the costs of all the steps. If in one step we partition set $A \cup B$ of items into two sets A and B, then the cost for this step would be $Cost(A, B) = |A \cup B| \times w(A, B)$ where $w(A, B)$ is the summation of all pairwise similarities between members of A and B. Considering this, taking the minimum cut as the partition at each step seems a reasonable choice, although we will see later that this would not give a good approximation ratio. A nice property of this objective function is its modularity. More precisely, suppose u is an internal node in tree T. If we replace T_u, the subtree rooted at u, by another subtree T'_u containing the same set of items as leaves, and denote the new tree by T', then the change in the total cost of the tree is only the difference between the costs of T_u and T'_u: $Cost(T') = Cost(T) + Cost(T'_u) - Cost(T_u)$.

Now let us consider a complete graph where any pair of items i and j are similar to each other with $w_{i,j} = 1$ as the first canonical example. In this case any binary tree T with n leaves would have $Cost(T) = \frac{1}{3}(n^3 - n)$. Also, if we consider the line graph on n nodes in which only consecutive items has unit similarities and any other pair of items has zero similarities, then the best tree would be the balanced binary tree which has the cost equal to $O(n \log n)$. Another interesting fact about this objective function is that the problem of minimizing the cost is equivalent to the problem of maximizing it. And here is the reason: For any graph G with unit edge weights denote its complement by G^c. Then for each tree T the summation of $Cost(T)$ over G and $Cost(T)$ over G^c would be the same and equal to the cost of a complete graph: $\frac{1}{3}(n^3 - n)$.

To prove the **NP**-hardness, we can use a reduction from a variant of Not-All-Equal SAT where each clause has two or three literals and each variable appears exactly three times, once in a 3-clause and twice with opposite polarities in two of 2-clauses and the hard problem is to find out if there is any assignment to the variables such that all the variables in each clause are not the same.

Dasgupta's shown that the top-down heuristic, which takes the minimum sparsest cut (approximately) at each step, has approximation factor of $O(\alpha \log n)$ [32], where α is the best approximation ratio for minimum sparsest cut problem which is $O(\sqrt{\log n})$ [7]. (In a more recent work, that we will discuss later, Charikar and Chatziafratis [21] showed that the algorithm of [32] in fact has approximation ratio of $O(\alpha)$.)

The intuition why the sparsest cut is a good option comes from the alternative interpretation of the cost function where we assigned a cost to each step. Recall that the cost of splitting $A \cup B$ into A and B would be $|A \cup B| \times w(A, B)$. To reduce the multiplier on subsequent steps that would be a good choice to shrink the set of items as much as we can, at each step. More precisely for a random item, the expected amount by which its cluster shrinks after this step would be:

$$\frac{|A|}{|A \cup B|} \cdot |B| + \frac{|B|}{|A \cup B|} \cdot |A| = \frac{2|A||B|}{|A \cup B|}$$

This means that a natural heuristic algorithm would be to choose a cut with maximum shrinkage per unit cost which is the minimum sparsest cut:

$$\frac{w(A, B)}{|A||B|}$$

We will provide some intuition on why this algorithm has a good approximation ratio later when we are discussing more recent works.

Later Roy and Pokutta [69] improved the previous result by giving an LP-based $O(\log n)$-approximation algorithm for the same objective function. They interpret the tree T as inducing an ultrametric d_T on the nodes as follows:

$$d_T(i, j) = T_{i,j} - 1$$

This is actually an ultrametric because $d_T(i, j) = 0$ if and only if $i = j$, and for any three nodes i, j, k we have $d_T(i, j) \leq max\{d_T(i, k), d_T(j, k)\}$. They introduced the notion of *non-trivial* ultrametrics as below and they are precisely the ultrametrics that are induced by any tree decompositions of the items corresponding to Dasgupta's cost function. Ultrametric d is non-trivial if:

- For every non-empty set of items S, there is a pair of items $i, j \in S$ such that $d(i,j) \geq |S| - 1$.
- For every $1 \leq t \leq n-1$, and every equivalence class S_t of the items under the relation $i \sim j$ if and only if $d(i,j) \leq t$, any pair of items $i, j \in S_t$ have $d(i,j) \leq |S_t| - 1$.

They proved two other properties of non-trivial ultrametrics. First, For every $1 \leq t \leq n-1$, and every equivalence class S_t of the items under the relation $i \sim j$ if and only if $d(i,j) \leq t$, restriction of d to S_t would be a non-trivial ultrametric on S_t. Second, the range of any non-trivial ultrametric on a set with cardinality n is contained in the set $\{0, 1, ..., n-1\}$.

They utilize these properties to conclude that for any non-trivial ultrametric d on the set of items, there is a corresponding hierarchical clustering T such that for any pair of items i, j we have $d(i,j) = T_{i,j} - 1$. Moreover this tree could be constructed using d in $O(n^3)$. This means that we can minimize our objective function over non-trivial ultrametrics with range contained in $\{0, 1, ..., n-1\}$ instead of trees.

They formulated the problem as an Integer Linear Program motivated by [73] and introduced variables x_{ij}^t for each pair i, j of items and $0 \leq t \leq n-1$ which is 1 if and only if $d(i,j) \geq t$. The objective function could be written as follow:

$$Cost(d) = \sum_{i,j \in [n]} \big(w_{i,j} \times \sum_{t=1}^{n-1} x_{ij}^t\big)$$

Roy and Pokutta [69] finally managed to relax the variables and then round it to have an $O(\log n)$-approximation for the problem which was considered an improvement at that time. (As we mentioned before, the simple heuristic sparsest cut algorithm was actually better than this with approximation ratio of $O(\sqrt{\log n})$ [21])

Roy and Pokutta [69] also provided some hardness results for the problem. Assuming the Small Set Expansion Hypothesis [67], the objective function introduced by Dasgupta [32] is hard to approximate to within any constant factor. In addition, they provided some experimental results that compares their algorithm with the current known algorithms including Average Linkage, Single Linkage, Complete Linkage, Ward's Method and bisecting k-means. Their results show that their algorithm often gives better pruning compared to the other standard clustering algorithms with respect to a notion of error they defined.

One year later, Charikar and Chatziafratis [21] provided a similar linear programming relaxation leading to an LP-based $O(\log n)$ approximation algorithm and the same constant factor inapproximability results for this problem simultaneously. In addition they presented a new spreading metric SDP relaxation and proved the integrality gap of $O(\sqrt{\log n})$ for it. In their analysis of that simple top-down Recursive Sparsest Cut (RSC), they managed to drop the $\log n$ factor and showed that the same algorithm actually yields an $O(\alpha)$ approximation.

Here is a brief intuition for the unweighted case: For each $0 \leq t \leq n$, let us O_t be the set of all maximal clusters of size at most t in optimal solution. Also, denote the set of edges that are cut in O_t (has endpoints in two different clusters) by E_t. Also, set $E_0 = E_1 = E$. The first observation would be this:

$$OPT = \sum_{t=0}^{n-1} |E_t|$$

The reason is that, for each edge $(i, j) \in E$, suppose that the size of the minimal cluster in optimal solution containing both i and j is r. Then the contribution of this edge to OPT is r. Also, $(i, j) \in E_t$ for all $t \leq r - 1$. Hence the contribution of this edge to the right-hand side is also r. Using this observation, we can have this lower bound for the optimal solution:

$$2OPT = 2\sum_{t=0}^{n-1} |E_t| \geq \sum_{t=0}^{n} |E_{\lfloor t/2 \rfloor}|$$

Now consider a cluster A with cardinality r in the solution produced by RSC algorithm and suppose it is partitioned into B_1 and B_2 with cardinalities s and $r-s$, respectively, such that $s \leq r/2$. Now consider $O_{\lfloor r/2 \rfloor}$ which contains the maximal clusters of size at most $r/2$ in the optimal solution, restricted to A. Suppose $A_1, ..., A_k$ are the induced inside the cluster A by $O_{\lfloor r/2 \rfloor} \cap A$ where $|A_i| = c_i|A|$ and $c_i \leq 1/2$ (because of the choice of $r/2$). Partition (B_1, B_2) is an α approximation for the sparsest cut problem and so we can say this:

$$\frac{|E(B_1, B_2)|}{s(r-s)} \leq \alpha \times \min_i \frac{|E(A_i, A \setminus A_i)|}{|A_i||A \setminus A_i|} = \alpha \times \min_i \frac{|E(A_i, A \setminus A_i)|}{c_i(1-c_i)r^2}$$

Then we can use this simple inequality:

$$\min_i \frac{a_i}{b_i} \leq \frac{\sum_i a_i}{\sum_i b_i}$$

By applying this to the previous inequality we have:

$$\frac{|E(B_1, B_2)|}{s(r-s)} \leq \alpha \times \frac{\sum_i |E(A_i, A \setminus A_i)|}{\sum_i c_i(1-c_i)r^2} \leq \alpha \times \frac{4}{r^2} \cdot |E_{\lfloor r/2 \rfloor} \cap A|$$

$$\Rightarrow \quad Cost(A) = r|E(B_1, B_2)| \leq 4\alpha \times s|E_{\lfloor r/2 \rfloor} \cap A| \leq 4\alpha \sum_{t=r-s+1}^{r} |E_{\lfloor t/2 \rfloor} \cap A|$$

Last inequality easily follows from the fact that $|E_t \cap A| \leq |E_{t-1} \cap A|$ by the definition of E_t and E_{t-1}. Now let us fix a $1 \leq t \leq n-1$. We want to know for which clusters A term $|E_{\lfloor t/2 \rfloor} \cap A|$ appears in $Cost(A)$. This could only be the case if A is a minimal cluster with $|A| \geq t > |B_2| = r - s \geq |B_1| = s$. Because of this minimality, the set of all such clusters would form a disjoint partition of all the items. Hence, $E_{\lfloor t/2 \rfloor} \cap A$ for all clusters A would form a partition of $E_{\lfloor t/2 \rfloor}$ and we can say:

$$\sum_A \sum_{t=r-s+1}^{r} |E_{\lfloor t/2 \rfloor} \cap A| \leq \sum_{t=0}^{n} |E_{\lfloor t/2 \rfloor}|$$

Now we can put together this upper bound and the lower bound for the optimal solution to have this:

$$Cost(RSC) = \sum_A Cost(A) \leq \sum_A 4\alpha \sum_{t=r-s+1}^{r} |E_{\lfloor t/2 \rfloor} \cap A| \leq 4\alpha \sum_{t=0}^{n} |E_{\lfloor t/2 \rfloor}| \leq 8\alpha OPT$$

This means that the algorithm is actually an 8α-approximation.

Cohen-Addad et al. [30] considered a fairly general random graph model for hierarchical clustering, called the hierarchical stochastic block model (HSBM), and showed that in certain regimes the SVD approach of McSherry [60] combined with specific linkage methods results in a clustering that give an $O(1)$ approximation to Dasgupta's cost function.

Moseley and Wang [63] considered the dual of Dasgupta's objective function for similarity-based graphs, where the objective is to maximize the following formula:

$$Rev_{Dual}(T) = \sum_{i,j \in [n]} \big(w_{i,j} \times (n - |T_{i,j}|) \big) \tag{4.3}$$

For each tree T we have $Cost(T) + Rev_{Dual}(T) = nW$ where W is the summation of the weights of similarity over all pairs of the items which is not dependent to the structure of the tree. This means that the optimum solution for both objective functions is the same. They showed that the classic average-linkage algorithm as well as the random top-down partitioning algorithm have approximation ratio 1/3 and provided a simple top-down local search algorithm that gives a $(\frac{1}{3} - \epsilon)$-approximation.

Later Charikar, Chatziafratis and Niazadeh [22] proved that the average-linkage algorithm is tight for both objective functions (1.5) and (4.3). They also gave two top-down algorithms to beat the average-linkage ratios. More specifically for maximizing $Rev_{Dual}(T)$ (4.3), they provided an SDP-based algorithm which has approximation ratio 0.336379 (slightly better than 1/3). For the maximizing dissimilarity-based graphs (1.5), they gave a top-down algorithm with a factor 0.667078 approximation (slightly better than 2/3). We will go through this algorithm in more details shortly as one of our results is to improve their approach.

More recently, Ahmadian et al. [3] provided a 0.4246-approximation algorithm for maximizing $Rev_{Dual}(T)$. What they do is to detect the cases where average linkage is not good and show that in those cases the maximum uncut bisection would gain a good fraction of the objective of the optimum solution in the very first step. So, by taking the better of the two of average-linkage and maximum uncut bisection (if we can solve it optimally in polynomial time) in the first step and average linkage for the remaining steps the approximation ratio would be 4/9. But the best known algorithm for maximum uncut bisection has approximation ratio $\rho = 0.8776$ [8], so the ratio of their algorithm decreases from $4/9$ to 0.4246 which is still much better than the previous best 0.3363-approximation of [22]. They also complemented their positive results by providing the **APX**-hardness (even for 0-1 similarities), under the Small Set Expansion hypothesis [67].

More recently, Alon et al. [5] proved that the algorithm of [3] is actually giving a $2\rho/3 = 0.585$-approximation by proving the existence of a better maximum uncut bisection. This is considered the third improvement over the Average-Linkage for revenue maximization of similarity-based Hierarchical Clustering (4.3), while the best algorithm for the revenue maximization of dissimilarity-based version (1.5) is still only slightly better than the average linkage (0.667 vs 2/3).

Chatziafratis et al. [25] considered a version of the problem where we have some prior information about the data that imposes constraints on the clustering hierarchy and provided provable approximation guarantees for two simple top-down algorithms on similarity-based graphs. More recently, Bakkelund [9] considered order preserving hierarchical agglomerative clustering which is a method for hierarchical clustering of directed acyclic graphs and other strictly partially ordered data that preserves the data structure.

Emamjomeh-Zadeh and Kempe [36] considered adaptive Hierarchical Clustering using the notion of ordinal queries, where each ordinal query consists of a set of three elements, and the response to a query reveals the two elements (among the three elements in the query) which are "closer" to each other. They studied active learning of a hierarchical clustering using only ordinal queries and focused on minimizing the number of queries even in the presence of noise.

Wang and Wang [76] suggested that Dasgupta's cost function is only effective in differentiating a good HC-tree from a bad one for a fixed graph, But the value of the cost function does not reflect how well an input similarity graph is consistent with a hierarchical structure and present a new cost function, which is based on Dasgupta's cost function but gives a cost between 0 and 1 to each tree.

Charikar et al. [23] were the first one to consider Hierarchical Clustering for Euclidean data and showed an improvement is possible for similarity-based graphs on objective (4.3). Later Wang and Moseley [77] considered objective (1.5) for Euclidean data and showed that every tree is a 1/2-approximation if the distances form a metric and developed a new global objective for hierarchical clustering in Euclidean space and proved that the optimal 2-means solution results in a constant approximation for their objective.

Hogemo et al. [46] considered the Hierarchical Clustering of unweighted graphs and introduced a proof technique, called the normalization procedure, that takes any such clustering of a graph G and iteratively improves it until a desired target clustering of G is reached. More recently Vainstein et al. [74] proved structural lemmas for both objectives (4.3) and (1.5) allowing to convert any HC tree to a tree with constant number of internal nodes while incurring an arbitrarily small loss. They managed to obtain approximations arbitrarily close to 1, if not all weights are small (i.e., there exist constants ϵ and δ such that the fraction of weights smaller than δ, is at most $1-\epsilon$).

Chehreghani [26] proposed a hierarchical correlation clustering method that extends the well-known correlation clustering to produce hierarchical clusters. Later Vainstein et al. [74] provided a 0.4767-approximation and presented nearly optimal approximations for complementary similarity/dissimilarity weights. There are also some other works including the ones trying to reduce the time of the current algorithms [62] and [33], and those introducing Fair Hierarchical Clustering [4] and Online Hierarchical Clustering [61].

4.2 Maximizing $Rev(T)$ in Dissimilarity-Based Graphs

For ease of exposition, we first present an algorithm and prove it has approximation 0.6929. Then we show how running the same algorithm with different parameters and taking the best solution of all we obtain a 0.71604-approximation. Our algorithm for maximizing $Rev(T)$ (objective function (1.5)) builds upon the algorithm of Charikar et al. [22] which has approximation ratio 0.667078, slightly better than random partition at each step, which has ratio 2/3.

As explained in [22], one can see that the top-level cuts of the tree, i.e. those corresponding to clusters closer to the root are making the large portion of the optimum value. Therefore, it seems reasonable, in a top-down approach, to use larger cut sizes at each step. So, this suggests a simple algorithm: at each step try to find a max-cut (or approximate max-cut). However, this "recursive max-cut" fails. As shown in [22], for a graph of n vertices with a clique of size ϵn where the rest of the edges have weight zero, the optimum solution "peels off" vertices of the implanted clique one by one (i.e. in initial steps each vertex of the clique is separated from the rest), and this obtains an objective value of at least $n(1-\epsilon)W$, where W is the sum of all edge weights. But the recursive max-cut (even if we find the optimum max-cut in each step) will have ratio at most $\frac{2+\epsilon}{3}nW$. Thus, this "recursive max-cut" is not going to perform better than the trivial random partitioning at each step.

Inspired by this, [22] suggested an algorithm that initially will peel off vertices with high (weighted) degree one by one (depending on a predetermined threshold) and after that will use a max-cut in one step to partition the remaining cluster into two. From there on, we can assume we use the random partitioning. This is the "peel-off first, max-cut next" algorithm of [22]. Note that the upper bound used so far in previous works for optimum is nW. Intuitively, if the optimum value is close to this lower bound then there must be a good max-cut, and if the optimum is bounded away from this then random partition performs better than 2/3. They showed that the better of this "peel-off first, max-cut next" algorithm and the random partitioning algorithm has approximation ratio of 0.667078 for maximizing $Rev(T)$ in dissimilarity-based graphs, which slightly beats the 2/3-approximation of simply doing random partitioning.

Our algorithm is based on similar ideas [22] but has more steps added into it to improve the bound. Our basic algorithm takes the better of the two of "Random Partitioning" algorithm and our "Peel-Off first, Max-Cut or Random Next" algorithm. This is to make sure in the cases where the revenue of the optimum solution is too far from nW, the random partitioning algorithm would give us a good enough approximation and we can focus on the case where the revenue of the optimum, which we denote by OPT, is close to nW.

A key difference between our "Peel-Off first, Max-Cut or Random Next" algorithm and algorithm of [22], which is "Peel-Off first Max-Cut Next" algorithm, is that after the Peel Off in the first phase we then choose the better of the two of "Random Partitioning" and "Max Cut" algorithms. Actually, by looking at the fraction of W which is peeled off in the first phase, we can decide which one of the "Random Partitioning" or "Max Cut" would be good enough for the second phase and only go ahead with that one. We have also generalized their analysis by considering a few more parameters to finally improve the approximation ratio from 0.667078 to 0.6929.

In our algorithm, we first set a parameter $\gamma \geq 1$ and start the Peel-Off process. More precisely we define W_v for each remaining vertex as the current total of the weights of the edges incident to v and remove vertices with $W_v \geq \gamma \frac{2W}{n}$ and all their edges, where W is the total of the weight of all the edges (not only the remaining edges) and n is the total number of vertices. So, W_v is a dynamic value (this is different from what [22] do as they compare the initial W_v with the threshold value). After removing one vertex (which is always the one with the largest W_v) we remove all the edges incident to it and update W_v for the remaining vertices accordingly. Note that, we do not update W and n and these two are fixed (to the initial values) throughout the entire peel-off process. So our Peel-Off process is a bit different than the one used in [22] as we need to update W_v for the remaining vertices after removing each vertex.

After we reach a state where all the remaining vertices have $W_v < \gamma \frac{2W}{n}$, we then look at the fraction of the W which is peeled off. Let us denote all the peeled off vertices by V_R and call them "Red" vertices (this is similar to the terminology used by [22]). We also denote all the edges incident to at least one red vertex, those that are removed from the graph after the peel off phase, by red edges, E_R, and denote the total weight of all the red edges by W_R and define $R = W_R/W$ as the weighted fraction of the red edges. We also call the vertices and edges that are not red, as blue vertices and edges, respectively: $V_B = V \setminus V_R$ and $E_B = E \setminus E_B$.

At this stage of our algorithm if R is greater than a pre-defined parameter $0 < R^* < 1/2$ (to be chosen later) this means that a good fraction of the vertices are peeled-off and we continue the algorithm by doing the random partitioning. Otherwise, this means that remaining graph is pretty dense, and we can prove there should be a big cut. So, if $R \leq R^*$ we continue the algorithm by doing an approximated max-cut. After max-cut we do the rest using random partitioning. Formally we propose "Peel-off First, Max Cut or Random Next" (Algorithm 1) and prove Theorem 12.

Algorithm 1 Peel-off First, Max Cut or Random Next

```
Input: $G = (V, E)$, dissimilarity weights$\{w_{i,j}\}_{(i,j)\in E}$, and parameters $\gamma \geq 1$ and $0 < R^* < 1/2$
  define $W = \sum_{(v,u)\in E} w_{v,u}$ and $n = |V|$.
  Initialize hierarchical clustering tree $T \leftarrow \emptyset$.
  Initialize $V_B \leftarrow V$ and $E_B \leftarrow E$.
  $While$ there exists a vertex $v \in V_B$ with $W_v = \sum_{u\in V_B:(v,u)\in E_B} w_{v,u} \geq \gamma\frac{2W}{n}$
    Choose $v^* \in V_B$ with the largest $W_v$.
    Update $T$ by adding the cut $(\{v^*\}, V_B \setminus \{v^*\})$.
    Update $V_B$ by removing $v^*$; $V_B \leftarrow V_B \setminus \{v^*\}$.
    Update $E_B$ by removing all the edges incident to $v^*$; $E_B \leftarrow E_B \setminus \{e \in E_B : e \text{ incident to } v^*\}$.
  define $W_R = W - \sum_{(v,u)\in E_B} w_{v,u}$ and $R = \frac{W_R}{W}$.
  $IF\ R > R^*$
    Recursively run Random Partitioning Algorithm on $V_B$ and update $T$.
  $ELSE$
    Run approximate Max-Cut [41] on $G_B = (V_B, E_B)$
    Let the resulting cut be $(V_L, V_R)$ and update $T$ by adding this cut.
    Recursively run Random Partitioning Algorithm on $V_L$ and $V_R$ and update $T$.
  $RETURN\ T$
```

Theorem 12. *For Hierarchical Clustering on dissimilarity-based graphs, there exists a choice of $\gamma \geq 1$ and $0 < R^* < 1/2$ such that the better of Algorithm 1 and "Random Partitioning" algorithm would be an α-approximation algorithm to maximize Rev(T), where $\alpha = 0.6929$.*

4.2.1 Proof of Theorem 12

To prove Theorem 12, we first need to have some more definitions. First recall that in an instance with n vertices and the total weights of dissimilarities of W, the random partitioning algorithm always has revenue at least $2nW/3$ [25].

In the cases where OPT is too far from nW the random partitioning algorithm would be good enough. More precisely we consider $OPT = (1-\epsilon)nW$, and this defines ϵ, which is not known to us. If $\epsilon \geq \frac{\alpha-2/3}{\alpha}$, then $OPT \leq \frac{2}{3\alpha}nW$ and the random partitioning algorithm would be an α-approximation. So, we assume $\epsilon < \frac{\alpha-2/3}{\alpha}$ and prove that there exist choices of γ and R^* which make Algorithm 1 an α-approximation for $\alpha = 0.6929$.

Lemma 13. *Let the number of vertices that are peeled off after the first phase of our algorithm be $|V_R| = n\ell$. Then $\ell \leq \frac{R}{2\gamma}$*

Proof. Recall that only those vertices that have $W_v \geq \gamma\frac{2W}{n}$ at the time of the peel-off are peeled off in the first phase. So, we can say that W_R, which is the summation of W_v of all the peeled off vertices, is at least $n\ell \cdot \gamma\frac{2W}{n}$. This means $W_R \geq 2\gamma\ell W$. □

Now assume ALG_P is the contribution of all the red edges (of our algorithm) to the final objective revenue $Rev(T)$. This is the revenue which is obtained in the peel-off phase. The following lemma is a stronger version of Lemma 5.1 of [22].

Lemma 14. $ALG_P \geq (1-\ell/2)nW_R$

Proof. Suppose $V_R = \{v_1, v_2, ..., v_{n\ell}\}$ where v_i is the i'th vertex which is peeled off. Recall that at each step of the peel-off phase of the algorithm we choose the vertex with the largest W_v. Thus $W_{v_1} \geq W_{v_2} \geq ... \geq W_{v_{n\ell}} \geq \gamma\frac{2W}{n}$. Now consider the revenue which is obtained in the peel-off phase. While removing v_i the size of the tree is $n-i+1$, so we have:

$$\begin{aligned}
ALG_P &= nW_{v_1} + (n-1)W_{v_2} + ... + (n - n\ell + 1)W_{v_{n\ell}} \\
&\geq \frac{\left(\sum_{i=n-n\ell+1}^{n} i\right) \cdot \left(\sum_{j=1}^{n\ell} W_{v_j}\right)}{n\ell} \\
&= \left(\frac{n\ell \cdot (n + (n - n\ell + 1))/2}{n\ell}\right) \cdot W_R \\
&= (1 - \ell/2 + 1/n) \cdot nW_R \\
&\geq (1 - \ell/2)nW_R.
\end{aligned}$$

□

As we mentioned earlier, after the peel-off phase, the algorithm will look at the ratio $R = W_R/W$ and compares it with the given parameter R^*. If $R > R^*$, the second phase will be to do the random partitioning and no max-cut is needed. Otherwise, if $R \leq R^*$, then the algorithm will use Goemans and Williamson's algorithm for Max-Cut [41] on the remaining graph (blue vertices and edges) to do one partition (the rest can be done in any arbitrary manner, say random partition). So, we separate our analysis in those two cases and prove necessary lemmas and theorems in each case.

Case 1: $R > R^*$

Let us denote the total revenue obtained in the random partitioning phase of the algorithm by ALG_R. Then the total revenue of our algorithm would be at least $ALG_P + ALG_R$. For this case of $R > R^*$ we prove the following theorem:

Theorem 13. *If $R > R^*$: $ALG_P + ALG_R \geq nW \cdot (\frac{2}{3} + \frac{\gamma-1}{3\gamma} \cdot R^* + \frac{1}{12\gamma} \cdot R^{*2})$.*

This means that the ratio of our algorithm in this case is at least $(\frac{2}{3} + \frac{\gamma-1}{3\gamma} \cdot R^* + \frac{1}{12\gamma} \cdot R^{*2})$. Note that to have this ratio, we are actually comparing our revenue with nW (and not OPT which might in fact be smaller) and if we somehow find a better upper-bound for OPT, comparing our revenue with OPT might prove a better ratio for our algorithm. To prove this Theorem, we first recall that in an instance with n vertices and the total weights of dissimilarities of W, random partitioning algorithm has revenue at least $2nW/3$ [25].

Lemma 15. $ALG_R \geq \frac{2}{3}(1 - \ell)n(W - W_R)$.

Proof. This follows easily from the observation mentioned above since after the peel-off phase the number of remaining vertices is $|V_B| = n - n\ell = (1-\ell)n$ and the total weights of the remaining edges is $W_B = W - W_R$. $\square$

The rest of the proof of Theorem 13 will be done by simply using Lemmas 13, 14, 15, and the fact that $R > R^*$. Combining Lemmas 14 and 15 we have:

$$\begin{aligned} ALG_P + ALG_R &\geq (1-\ell/2)nW_R + \frac{2}{3}(1-\ell)n(W - W_R) \\ \Rightarrow \frac{ALG_P + ALG_R}{nW} &\geq (1-\ell/2)R + \frac{2}{3}(1-\ell)(1-R). \end{aligned}$$

The RHS is decreasing with ℓ increasing, so using Lemma 13, we have:

$$\Rightarrow \frac{ALG_P + ALG_R}{nW} \geq (1-\frac{R}{4\gamma})R + \frac{2}{3}(1-\frac{R}{2\gamma})(1-R) = (\frac{2}{3} + \frac{\gamma-1}{3\gamma}\cdot R + \frac{1}{12\gamma}\cdot R^2).$$

This bound is increasing with R, so considering the fact that $R > R^*$, it implies Theorem 13.

Let us define function $F(R) = \left(1 - \frac{R}{4\gamma}\right)R + \frac{2}{3}\left(1 - \frac{R}{2\gamma}\right)(1-R)$ and note that in Theorem 13 we just proved that if $R > R^*$ then $ALG_P + ALG_R \geq nW \cdot F(R^*)$. We will use this bound later.

Case 2: $R \leq R^*$

Like the other case, let us denote the total revenue obtained in the max-cut phase of the algorithm by ALG_C. Then the total revenue of our algorithm would be at least $ALG_P + ALG_C$. Note that we are not considering the revenue obtained after the max-cut phase, which uses random partitioning (as we do not have a good bound on the value/weight of the edges left after performing the max-cut). For this case of $R \leq R^*$ we prove the following theorem:

Theorem 14. *Suppose $R \leq R^*$ and the following conditions for ϵ, R^*, and γ hold (where α_{GW} is the ratio of approximation for max-cut [41]):*

$$R^* < 1 - \frac{6\alpha_{GW}}{3\alpha_{GW} - 2}\cdot\epsilon. \tag{4.4}$$

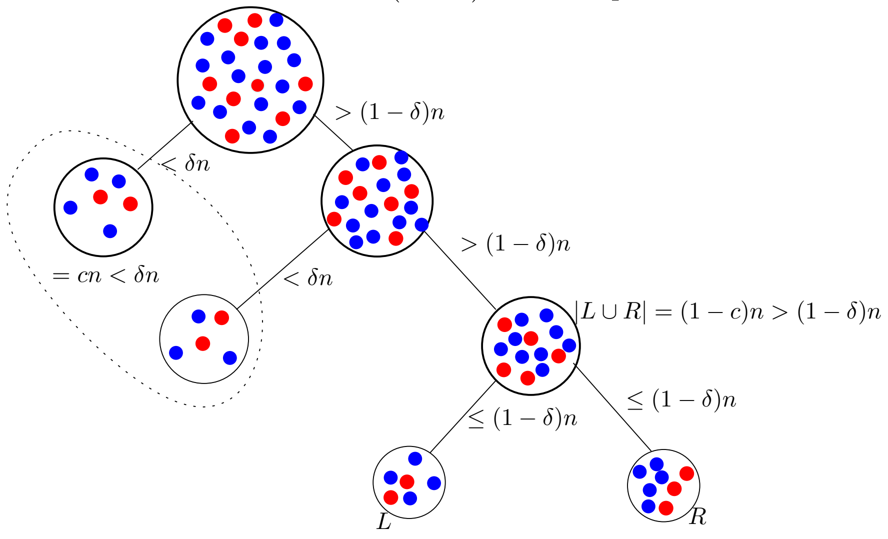
$$\gamma \leq (1-R^*)^2 \cdot \frac{3\alpha_{GW} - 2}{9\alpha_{GW}^2}\cdot\frac{1}{\epsilon}. \tag{4.5}$$

$$\frac{1}{3} \geq R^* \cdot \left(\frac{\alpha_{GW}}{2} - \frac{1}{6}\right) + \gamma\cdot(1-\alpha_{GW}). \tag{4.6}$$

Then $ALG_P + ALG_C \geq nW \cdot F(R^*)$.

This means that the ratio of our algorithm in this case (like the other case) is at least $F(R^*)$ if conditions (4.4), (4.5), and (4.6) are met.

We continue by looking at the layered structure of the optimum tree similar to the one done in [22]. Fix parameter $\delta < 1/2$ (to be specified later) and imagine we start from the root of the optimum tree. We are looking for the smallest (i.e. deepest) cluster of the tree with size more than $n(1-\delta)$. This is a cluster where itself and all its ancestors have size more than $n(1-\delta)$ but both its children are smaller. Consider each time the optimum tree performs a partition of a cluster into two. Since $\delta < 1/2$, either both of these clusters must be of size at most $n(1-\delta)$ (at which point we stop), or exactly one of them is smaller than $n(1-\delta)$ and the other is strictly larger than $n(1-\delta)$. In the latter case, we go down the bigger branch and keep doing this until both children have size at most $n(1-\delta)$. At this point we have found the smallest



Layered structure of optimum tree and red/blue vertices

cluster (internal node) of the tree with size more than $n(1-\delta)$, which then has two (children) clusters of size at most $n(1-\delta)$. We denote these two children clusters by L and R and we have: $|L| < n(1-\delta)$, $|R| < n(1-\delta)$, $|L \cup R| > n(1-\delta)$, and finally $|V \setminus (L \cup R)| < n\delta$. We define $c = |V \setminus (L \cup R)|/n$ and we know $0 \leq c < \delta$.

Now we partition the blue vertices (vertices that survived the peel off phase) into two groups. $V_{B-Cut} = V_B \cap (L \cup R)$ and $V_{B-Chain} = V_B \setminus (L \cup R)$. Note that V_{B-Cut} are those inside the smallest cluster (internal node) of the optimum tree with size more than $n(1-\delta)$ and $V_{B-Chain}$ are those outside it. We also define $W_{B-Chain}$ as the total weights of the blue edges incident to at least one vertex of $V_{B-Chain}$. Now W_B is partitioned into four part: $W_B = W_L + W_R + W_{L,R} + W_{B-Chain}$, where $W_{L,R}$ are those with one end in L and the other in R while W_L and W_R are those with both ends in L and R, respectively. Now we are ready for our first lemma.

Lemma 16. *$W_{B-Chain} \leq 2c\gamma W$.*

Proof. We know that $|V_{B-Chain}| \leq |V \setminus (L \cup R)| = cn$. We also know that each $v \in V_{B-Chain}$ has $W_v < \gamma \frac{2W}{n}$:

$$W_{B-Chain} \leq \sum_{v \in V_{B-Chain}} W_v < |V_B| \cdot \gamma \frac{2W}{n} = 2c\gamma W.$$

□

Now it iss time to find a cut in V_B with provably good size. This layered structure of the optimum tree would suggest the cut (L, R) with size $W_{L,R}$ and the following lemma will give us a lower-bound on its size:

Lemma 17. *$W_{L,R} \geq \frac{\delta W_B - (\epsilon + 2c\delta\gamma)W}{\delta - c}$.*

Proof. Consider the revenue of the optimum solution. By looking at the layered structure we just introduced we have:

$$OPT \leq n(W_R + W_{B-Chain}) + n(1-\delta)(W_L + W_R) + n(1-c)W_{L,R}.$$

This is because the size of the cluster when the optimum solution produces cut (L, R) is exactly $n(1-c)$ and after that the size of the following clusters could not be more than $n(1-\delta)$. Now we use $OPT = (1-\epsilon)nW$ and $W_L + W_R = W_B - W_{B-Chain} - W_{L,R}$ in the above inequality:

$$OPT = (1-\epsilon)nW \leq n(W_R + W_{B-Chain}) + n(1-\delta)(W_B - W_{B-Chain} - W_{L,R}) + n(1-c)W_{L,R}.$$

By dividing both sides by n and considering the fact that $W = W_R + W_B$ we have:

$$\Rightarrow W - \epsilon W \leq W - \delta(W_B - W_{B-Chain} - W_{L,R}) - cW_{L,R}.$$

$$\Rightarrow \delta(W_B - W_{B-Chain}) - \epsilon W \leq (\delta - c)W_{L,R}.$$

Considering the fact that c is strictly less than δ and using Lemma 16 this completes the proof of the lemma. □

The next lemma will provide a lower bound on ALG_C.

Lemma 18. $ALG_C \geq \alpha_{GW} \cdot (1-\ell)nW_{L,R}$.

Proof. Note that after the peel-off phase, we are going to use Goemans and Williamson's max-cut algorithm to find a cut in V_B whose size is $n - n\ell$. As we know there is a cut with size at least $W_{L,R}$, we can find one with size $\alpha_{GW}W_{L,R}$ and because there are $(1-\ell)n$ many blue vertices left after the first phase, the revenue we will have on the first step of the second phase will be at least $\alpha_{GW} \cdot (1-\ell)nW_{L,R}$. □

Note that if we can somehow find a lower-bound for the revenue which is obtained after this max-cut step, (perhaps using a bi-section instead of a max-cut) our ratio could be even better. We should point out that using a bi-section [3] obtained the improved ratio for the similarity-based version.

Considering the fact that $R = W_R/W$, $W = W_R + W_B$ and combining the previous two lemmas we have:

$$ALG_C \geq \alpha_{GW} \cdot (1-\ell)nW \left(\frac{\delta(1-R) - (\epsilon + 2c\delta\gamma))}{\delta - c} \right). \tag{4.7}$$

Now we want to set $\delta = \frac{3\alpha_{GW}}{3\alpha_{GW}-2} \cdot \frac{\epsilon}{1-R^*}$, and to be able to do that we need Condition (4.4). That is because we have to make sure $\delta < 1/2$ and Condition (4.4) will be enough to have that:

$$R^* < 1 - \frac{6\alpha_{GW}}{3\alpha_{GW}-2} \cdot \epsilon \quad \Rightarrow \quad \frac{6\alpha_{GW}}{3\alpha_{GW}-2} \cdot \epsilon < 1 - R^* \quad \Rightarrow \quad \frac{3\alpha_{GW}}{3\alpha_{GW}-2} \cdot \frac{\epsilon}{1-R^*} < 1/2$$

Now we apply $\delta = \frac{3\alpha_{GW}}{3\alpha_{GW}-2} \cdot \frac{\epsilon}{1-R^*}$ to Equation (4.7):

Corollary 3. *$ALG_C \geq \alpha_{GW}\cdot(1-\ell)nW\cdot G(c)$ where $G(c) = \left(\frac{(1-R)-(1-R^*)\frac{3\alpha_{GW}-2}{3\alpha_{GW}}-2c\gamma)}{1-c\frac{(1-R^*)}{\epsilon}\frac{3\alpha_{GW}-2}{3\alpha_{GW}}}\right)$.*

Note that $0 \leq c < \delta = \frac{3\alpha_{GW}}{3\alpha_{GW}-2}\cdot\frac{\epsilon}{1-R^*}$, and the following lemma would make sure the worst case for us is when $c = 0$ (see Appendix 4.4 for proof).

Lemma 19. *If Condition (4.5) holds, then $G(0) \leq G(c)$ for all $0 \leq c < \delta = \frac{3\alpha_{GW}}{3\alpha_{GW}-2}\cdot\frac{\epsilon}{1-R^*}$.*

Using this lemma and applying $c = 0$ to Corollary 3 we have:

$$ALG_C \geq \alpha_{GW}\cdot(1-\ell)nW\left((1-R)-(1-R^*)\frac{3\alpha_{GW}-2}{3\alpha_{GW}}\right). \tag{4.8}$$

Using this and Lemmas 13 and 14 we can conclude:

$$\frac{ALG_P + ALG_C}{nW} \geq \left(1-\frac{R}{4\gamma}\right)R+\alpha_{GW}\cdot\left(1-\frac{R}{2\gamma}\right)\left((1-R)-(1-R^*)\frac{3\alpha_{GW}-2}{3\alpha_{GW}}\right). \tag{4.9}$$

Let us define function $\tilde{F}(R) = \left(1-\frac{R}{4\gamma}\right)R+\alpha_{GW}\cdot\left(1-\frac{R}{2\gamma}\right)\left((1-R)-(1-R^*)\frac{3\alpha_{GW}-2}{3\alpha_{GW}}\right)$. Note that the equation above says $\frac{ALG_P+ALG_C}{nW} \geq \tilde{F}(R)$ and:

$$\begin{aligned}
\tilde{F}(R^*) &= \left(1-\frac{R^*}{4\gamma}\right)R^* + \alpha_{GW}\left(1-\frac{R^*}{2\gamma}\right)\left((1-R^*)-(1-R^*)\frac{3\alpha_{GW}-2}{3\alpha_{GW}}\right) \\
&= \left(1-\frac{R^*}{4\gamma}\right)R^* + \alpha_{GW}\left(1-\frac{R^*}{2\gamma}\right)(1-R^*)\left(1-\frac{3\alpha_{GW}-2}{3\alpha_{GW}}\right) \\
&= \left(1-\frac{R^*}{4\gamma}\right)R^* + \alpha_{GW}\left(1-\frac{R^*}{2\gamma}\right)(1-R^*)\frac{2}{3\alpha_{GW}} \\
&= \left(1-\frac{R^*}{4\gamma}\right)R^* + \frac{2}{3}\left(1-\frac{R^*}{2\gamma}\right)(1-R^*) \\
&= F(R^*)
\end{aligned}$$

So, the only thing we need to show to complete the proof of Theorem 14 is that $\tilde{F}(R)$ is decreasing with R increasing in interval $0 \leq R \leq R^*$ and the following lemma will prove this (see Appendix 4.4):

Lemma 20. *If Condition (4.6) holds, then $\tilde{F}(R^*) \leq \tilde{F}(R)$ for all $0 \leq R \leq R^*$.*

It is easy to see that Equation (4.9) together with Lemma 20 imply Theorem 14.

Using Theorems 13 and 14 to prove Theorem 12

To complete the proof of Theorem 12, we use Theorems 13 and 14 to conclude that if we can set γ and R^* such that Conditions (4.4), (4.5), and (4.6) are met, then the ratio of our algorithm would be at least $F(R^*)$:

$$F(R^*) = \left(1 - \frac{R^*}{4\gamma}\right) R^* + \frac{2}{3}\left(1 - \frac{R^*}{2\gamma}\right)(1 - R^*). \tag{4.10}$$

Recall that we can assume $\epsilon < \frac{\alpha - 2/3}{\alpha}$ as otherwise the random partitioning gives a better than α-approximation. If we fix a value for α, this condition implies a bound for ϵ, which then using Condition (4.5) gives a bound for γ: $\gamma = (1-R^*)^2 \cdot \frac{3\alpha_{GW}-2}{9\alpha_{GW}^2} \cdot \frac{\alpha}{\alpha-2/3}$ and this in turn implies the best value for R^* using the equation above for $F(R^*)$. Note that the ratio of our algorithm would be the minimum of α and $F(R^*)$. It is easy to verify that by having $\alpha = 0.6929$, we will get $R^* = 0.227617$ which maximizes $F(.)$ exactly at $F(R^*) = \alpha$; for this value γ will be set to 1.442042. Note that these values of α, R^* and γ will satisfy all the conditions of Theorem 14 and the ratio of algorithm will be 0.6929. This completes the proof of Theorem 12.

4.2.2 Improving the ratio to 0.71604

In this section we will show how to improve the ratio of our algorithm to 0.71604. A key observation from the past section is that while we use ϵ in our conditions, at the end we are comparing the revenue obtained by the algorithm with nW and not with $OPT = (1-\epsilon)nW$. In other words, the function $F(.)$ is obtained based on comparing the solution of the algorithm with nW (in both Theorems 13 and 14). This is because we do not have any lower-bound on ϵ and it really could be arbitrarily close to 0. However, when ϵ is close to 0, then our conditions, especially Condition (4.5), will let us choose better γ and R^*. We will take advantage of this to find a series of values for γ and R^*, such that if we run our algorithm with these parameters and take the better of all solutions then the ratio will be 0.71604.

Recall that to maximize the ratio of our algorithm we first set α and use our conditions to find the best R^* and γ to maximize $F(R^*)$ (using Equation (4.10)) and finally the ratio would be the minimum of α and $F(R^*)$. If $F(R^*) < \alpha$, it means that our initial choice of α was too high and vise versa. Suppose we set α to some value such that based on that value we obtain R_1^* and γ_1 that are maximizing $F(R_1^*)$ but $F(R_1^*) < \alpha$. As we mentioned earlier, since the actual ratio of our algorithm is $\frac{F(R_1^*)}{1-\epsilon}$ (since we assume $OPT = (1-\epsilon)nW$), there is an $\epsilon_2 < \epsilon$ where $\frac{F(R_1^*)}{1-\epsilon_2} = \alpha$ and for all values of ϵ where $\epsilon_2 \leq \epsilon \leq \epsilon_1$ (we define $\epsilon_1 = \frac{\alpha-2/3}{\alpha}$) the algorithm with parameters R_1^* and γ_1 is actually an α-approximation and all three conditions of Theorem 14 are met. But what happens if $\epsilon < \epsilon_2$? In this case the bounds of those three conditions are in fact better and we can set the parameters R^* and γ differently to obtain better ratios.

As an example, suppose that we want to improve the ratio to $\alpha = 0.7$. Using the bound $\epsilon \leq \epsilon_1 = \frac{\alpha-2/3}{\alpha} = 1/21$, we use Condition (4.5) and set $\gamma = (1-R^*)^2 \cdot \frac{3\alpha_{GW}-2}{9\alpha_{GW}^2} \cdot 21 = 1.9218297 \cdot (1-R^*)^2$ and find the best R^* to maximize $F(R^*)$ in Equation (4.10). It turns out the maximum of $F(.)$ is obtained at $R_1^* = 0.173114$ and $\gamma_1 = 1.314032$, with $F(R_1^*) = 0.682358$. This means that by running our algorithm with parameters R_1^* and γ_1 we will have the revenue at least $0.682358nW$. To make this an α-approximation (for $\alpha = 0.7$), we must have a lower-bound on ϵ. More precisely $0.682358nW \geq \alpha OPT = 0.7 \cdot (1-\epsilon)nW$ only if $\epsilon \geq \epsilon_2 = 0.02520285$. This means that if $\epsilon_2 \leq \epsilon \leq \epsilon_1$, then the revenue gained by our algorithm using parameters R_1^* and γ_1 is at least αOPT. Recall that if $\epsilon > \epsilon_1$, then random partitioning is an α-approximation. The only remaining case is if $\epsilon < \epsilon_2$, and in this case we can set $\gamma = (1-R^*)^2 \cdot \frac{3\alpha_{GW}-2}{9\alpha_{GW}^2} \cdot \frac{1}{\epsilon_2} = 3.6311637(1-R^*)^2$ and again find the best R^* to maximize $F(R^*)$. We must set $R_2^* = 0.316719$ and $\gamma_2 = 1.695292$ to maximize $F(R_2^*) = 0.714896$ which is even greater than α. This means that if $\epsilon < \epsilon_2$, then the revenue gained by our algorithm using parameters R_2^* and γ_2 is at least αOPT. Note that for γ_1, R_1^* and $\epsilon_1 = \frac{\alpha-2/3}{\alpha}$ and for $\gamma_2, R_2^*, \epsilon_2$ all three conditions of Theorem 14 are met. Because we do not know the exact value of ϵ, the only thing we need to do is to take the better of the following three:

1. Run Random Partitioning Algorithm all the way.
2. Run Algorithm 1 with parameters R^*_1 and γ_1.
3. Run Algorithm 1 with parameters R^*_2 and γ_2.

So, with only two sets of parameters for R^* and γ and running Algorithm 1 for each (and taking the better of the results as well as random partitioning) we can get a 0.7-approximation. It turns out using this approach and running the algorithm with several more parameters we can get slight improvement. More specifically, starting with $\alpha = 0.716$ we will find 83 triples of values $\gamma_i, R^*_i, \epsilon_i$ $(1 \leq i \leq 83)$ such that for each triple, the three conditions of Theorem 14 are met and for each pair of γ_i, R^*_i if we run Algorithm 1 with these parameters if the actual value of ϵ is between ϵ_{i+1} and ϵ_i (with the assumption of $\epsilon_{84} = 0$) then the revenue of the solution is at least αOPT. These 83 triples of $\gamma_i, R^*_i, \epsilon_i$ are obtained using a simple computer program and are listed in Table 4.1 in Section 4.5. By choosing $\alpha = 0.71604$ we will have 211 triples of $\gamma_i, R^*_i, \epsilon_i$ and ratio will be at least 0.71604.

4.3 Proof of Theorem 5

We discuss a method for bounding the cost of any top-down algorithm regarding this new objective function (1.6). When a top-down algorithm splits cluster $A \cup B$ into clusters A and B, the least common ancestor for any two pair of nodes $a \in A$ and $b \in B$ is determined. But for each two nodes $a, a' \in A$ (also for $b, b' \in B$), the distance between their least common ancestor and the root in the final tree is increased by 1. Given this, we define the cost of partitioning a cluster into two clusters A and B as the following formula:

$$\textit{split-cost}_H(A, B) = \sum_{a,a' \in A} w_{a,a'} + \sum_{b,b' \in B} w_{b,b'}.$$

Notice that the total cost $Cost_H(T)$ is exactly the sum of the *split-cost* over all internal nodes of tree T plus and additional $\sum_{i,j \in [n]} w_{i,j}$. We model the problem as a multi-step game. At each step we have to choose a cut (S_1, S_2) in G and remove its cutting edges $\delta(S_1) = \delta(S_2)$. The cost of step 0 is W and the cost of each step is the total weight of all the remaining edges. Then the total cost of the algorithm would be summation over all iterations. Considering this view, it would make sense to choose the maximum weighted cut at each step. One of our main results is to analyze this algorithm by showing that it has a constant approximation ratio. Note that this algorithm has a large approximation ratio when we want to minimize Dasgupta's objective function $Cost(T)$ or maximize its dual $Rev_{Dual}(T)$.

The method we use to prove Theorem 5 is inspired by the work done by Feige et al. [37]. We assume all the weights are integers and for each edge $(i,j) \in E$ we replace it with $w_{i,j}$ many unit weight edges to make the graph unweighted. Let OPT be the cost of the optimal tree and AMC be the cost of the approximated Max-Cut algorithm. For $i = 1, 2, ...$ let X_i^* and X_i be the sets of edges removed at step i by the optimal tree and approximated Max-Cut algorithm, respectively. Also, let R_i^* and R_i be the sets of edges remained after step i of the algorithms. We also set $R_0^* = R_0 = E$. Notice that for each $i \geq 0$ we have $R_i = E \setminus \cup_{j=1}^{i-1} X_i$, $R_i = X_{i+1} \cup R_{i+1}$ and $R_i = \cup_{j=i+1}^{n} X_j$ (same for R_i^*).

Now observe that we have:

$$OPT = \sum_{i=0}^{n} |R_i^*| = \sum_{i=1}^{n} i \cdot |X_i^*| \qquad \text{and} \qquad AMC = \sum_{i=0}^{n} |R_i| = \sum_{i=1}^{n} i \cdot |X_i|.$$

Now we define $p_1 = 0$ and for each $i \geq 2$ we define $p_i = \frac{|R_{i-1}|}{|X_i|}$ and for each $e \in E$ we set $p_e = p_i$ if $e \in X_i$, we then have:

$$\sum_{e \in E} p_e = \sum_{i=1}^{n} \sum_{e \in X_i} p_i = \sum_{i=1}^{n} |X_i| \cdot p_i = \sum_{i=2}^{n} |X_i| \cdot \frac{|R_{i-1}|}{|X_i|} = \sum_{i=1} |R_i| = AMC - |R_0|.$$

Observe that if you use an algorithm that chooses a cut that contains at least half of the edges at each step, including approximated maximum cut, then for each $e \in E \setminus X_1$ we have $p_e \leq 2$. Note that $p_e = p_1 = 0$ for each $e \in X_1$, so we have the following upper-bound for AMC:

$$AMC - |R_0| = \sum_{e \in E \setminus X_1} p_e \leq \sum_{e \in E \setminus X_1}^{n} 2 = 2|R_1| \qquad \Rightarrow \qquad AMC \leq |R_0| + 2|R_1|.$$

Now we consider two cases. For the first case we assume $|R_1^*| > \frac{2\alpha_{GW}-1}{2\alpha_{GW}} \cdot |R_0|$, and the second case is when $|R_1^*| \leq \frac{2\alpha_{GW}-1}{2\alpha_{GW}} \cdot |R_0|$, where α_{GW} is the ratio of approximate max-cut algorithm.

For the first case, note that X_1 is the cut the approximated maximum cut chooses in the first step and it has at least half of the edges, so $|R_1| \leq \frac{|E|}{2} = \frac{|R_0|}{2}$. If use the lower bound of $|R_0| + |R_1^*|$ for OPT, then we have:

$$\frac{AMC}{OPT} \leq \frac{|R_0| + 2|R_1|}{|R_0| + |R_1^*|} < \frac{|R_0| + |R_0|}{|R_0| + \frac{2\alpha_{GW}-1}{2\alpha_{GW}}|R_0|} = \frac{2}{1 + \frac{2\alpha_{GW}-1}{2\alpha_{GW}}} = \frac{4\alpha_{GW}}{4\alpha_{GW}-1}.$$

For the second case, note that X_1^* is a cut in G and X_1 has at least α_{GW} fraction of the maximum cut or any other cut including X_1^*, so:

$$|X_1| \geq \alpha_{GW} \cdot |X_1^*|.$$

So, for R_1 we have:

$$|R_1| = |R_0| - |X_1| \leq |R_0| - \alpha_{GW} \cdot |X_1^*| = |R_0| - \alpha_{GW} \cdot (|R_0| - |R_1^*|) = (1 - \alpha_{GW})|R_0| + \alpha_{GW} \cdot |R_1^*|.$$

Then again, we use the lower bound of $|R_0| + |R_1^*|$ for OPT to bound the ratio of the Approximated Max-Cut algorithm:

$$\frac{AMC}{OPT} \leq \frac{|R_0| + 2|R_1|}{|R_0| + |R_1^*|} \leq \frac{|R_0| + 2(1 - \alpha_{GW})|R_0| + 2\alpha_{GW} \cdot |R_1^*|}{|R_0| + |R_1^*|} = \frac{(3 - 2\alpha_{GW})|R_0| + 2\alpha_{GW} \cdot |R_1^*|}{|R_0| + |R_1^*|}.$$

Remember that in this case $|R_0| \geq \frac{2\alpha_{GW}}{2\alpha_{GW}-1} \cdot |R_1^*|$, so:

$$\frac{AMC}{OPT} \leq (3 - 2\alpha_{GW}) + \frac{(4\alpha_{GW} - 3) \cdot |R_1^*|}{|R_0| + |R_1^*|} \leq (3 - 2\alpha_{GW}) + \frac{(4\alpha_{GW} - 3) \cdot |R_1^*|}{\frac{2\alpha_{GW}}{2\alpha_{GW}-1} \cdot |R_1^*| + |R_1^*|}.$$

$$\Rightarrow \quad \frac{AMC}{OPT} \leq (3 - 2\alpha_{GW}) + \frac{(4\alpha_{GW} - 3)(2\alpha_{GW} - 1)}{(4\alpha_{GW} - 1)} = \frac{4\alpha_{GW}}{4\alpha_{GW} - 1}.$$

Thus, in either case, the cost of the solution returned by the recursively finding an α_{GW}-approximate max-cut is at most $\frac{4\alpha_{GW}}{4\alpha_{GW}-1}$ times the optimum. This completes the proof of Theorem 5.

4.4 Missing Proofs

Proof of Lemma 19. To prove this lemma, we need to show that $G(c)$ is increasing with c. And note that a function like $\frac{X-Yc}{1-Zc}$, where X, Y and Z are all independent of c, is ascending if and only if $XY \geq Z$. In $G(c)$ we have $X = (1 - R) - (1 - R^*)\frac{3\alpha_{GW}-2}{3\alpha_{GW}}$, $Y = 2\gamma$ and $Z = \frac{(1-R^*)}{\epsilon}\frac{3\alpha_{GW}-2}{3\alpha_{GW}}$. Now recall that we are in a case where $R \leq R^*$, so:

$$X \geq (1-R^*)-(1-R^*)\frac{3\alpha_{GW}-2}{3\alpha_{GW}} = (1-R^*)\cdot(1-\frac{3\alpha_{GW}-2}{3\alpha_{GW}}) = (1-R^*)\frac{2}{3\alpha_{GW}}.$$

Now using the definition of X, Y, Z and the bound above for X, Condition (4.5) implies $XZ \geq Y$, which completes the proof of the lemma. ■

Proof of Lemma 20. First we do some simplifications to $\tilde{F}(R)$:

$$\begin{aligned}
\tilde{F}(R) &= \left(1-\frac{R}{4\gamma}\right)R + \alpha_{GW}\left(1-\frac{R}{2\gamma}\right)\left((1-R)-(1-R^*)\frac{3\alpha_{GW}-2}{3\alpha_{GW}}\right) \\
&= \alpha_{GW}-(1-R^*)(\alpha_{GW}-2/3) - R\cdot\left(\alpha_{GW}+\frac{\alpha_{GW}}{2\gamma}-1-\frac{(1-R^*)(\alpha_{GW}-2/3)}{2\gamma}\right) \\
&+ R^2\cdot\left(\frac{\alpha_{GW}}{2\gamma}-\frac{1}{4\gamma}\right).
\end{aligned}$$

Let $X = \alpha_{GW}-(1-R^*)(\alpha_{GW}-2/3)$, $Y = \left(\alpha_{GW}+\frac{\alpha_{GW}}{2\gamma}-1-\frac{(1-R^*)(\alpha_{GW}-2/3)}{2\gamma}\right)$, and $Z = \left(\frac{\alpha_{GW}}{2\gamma}-\frac{1}{4\gamma}\right)$; then $\tilde{F}(R) = X - YR + ZR^2$. Observe that since $\gamma > 0$, Z is positive. Also, we will soon show that Y is positive too. Thus, the minimum of $\tilde{F}(R)$ is at $R = \frac{Y}{2Z}$. If we show that $R^* \leq \frac{Y}{2Z}$ then we have shown that $\tilde{F}(R)$ is totally descending in the interval $0 \leq R \leq R^*$. So, it is enough to show:

$$R^* \leq \frac{\alpha_{GW}+\frac{\alpha_{GW}}{2\gamma}-1-\frac{(1-R^*)(\alpha_{GW}-2/3)}{2\gamma}}{2(\frac{\alpha_{GW}}{2\gamma}-\frac{1}{4\gamma})}.$$

Or equivalently we need to have:

$$R^*(\alpha_{GW}-1/2) \leq R^*(\alpha_{GW}/2-1/3)+1/3-\gamma(1-\alpha_{GW}).$$

This inequality is exactly what we have in Condition (4.6). Also, since this condition implies that $R^* < \frac{Y}{2Z}$ and we know that $R^* > 0$, this also means $Y > 0$. Thus, the minimum of $\tilde{F}(R)$ is at $R = R^*$ and this completes the proof of the lemma. ■

4.5 R^* and γ Values to have a 0.716-approximation

Table 4.1: Values of parameters R^*, γ and ϵ to run Algorithm 1 and take the best and improve the approximation ratio to 0.716

R^*	γ	$F(R^*)$	ϵ
$R_1^* = 0.07852$	$\gamma_1 = 1.1278206$	$F(R_1^*) = 0.670089$	$\epsilon_1 = 0.0689014$
$R_2^* = 0.09761$	$\gamma_2 = 1.1621875$	$F(R_2^*) = 0.67189$	$\epsilon_2 = 0.0641222$
$R_3^* = 0.10809$	$\gamma_3 = 1.1817295$	$F(R_3^*) = 0.673031$	$\epsilon_3 = 0.0616056$
$R_4^* = 0.11489$	$\gamma_4 = 1.1946796$	$F(R_4^*) = 0.673828$	$\epsilon_4 = 0.0600121$
$R_5^* = 0.11973$	$\gamma_5 = 1.2039729$	$F(R_5^*) = 0.67442$	$\epsilon_5 = 0.0588994$
$R_6^* = 0.12337$	$\gamma_6 = 1.2110441$	$F(R_6^*) = 0.67488$	$\epsilon_6 = 0.0580723$
$R_7^* = 0.12622$	$\gamma_7 = 1.216645$	$F(R_7^*) = 0.67525$	$\epsilon_7 = 0.0574297$
$R_8^* = 0.12853$	$\gamma_8 = 1.2211909$	$F(R_8^*) = 0.675554$	$\epsilon_8 = 0.0569138$
$R_9^* = 0.13045$	$\gamma_9 = 1.2249629$	$F(R_9^*) = 0.67581$	$\epsilon_9 = 0.0564888$
$R_{10}^* = 0.13206$	$\gamma_{10} = 1.2282025$	$F(R_{10}^*) = 0.676029$	$\epsilon_{10} = 0.0561313$
$R_{11}^* = 0.13346$	$\gamma_{11} = 1.2309501$	$F(R_{11}^*) = 0.676219$	$\epsilon_{11} = 0.0558255$
$R_{12}^* = 0.13467$	$\gamma_{12} = 1.2333801$	$F(R_{12}^*) = 0.676386$	$\epsilon_{12} = 0.05556$
$R_{13}^* = 0.13574$	$\gamma_{13} = 1.2355205$	$F(R_{13}^*) = 0.676535$	$\epsilon_{13} = 0.0553267$
$R_{14}^* = 0.13669$	$\gamma_{14} = 1.2374426$	$F(R_{14}^*) = 0.676668$	$\epsilon_{14} = 0.0551194$
$R_{15}^* = 0.13755$	$\gamma_{15} = 1.2391589$	$F(R_{15}^*) = 0.676788$	$\epsilon_{15} = 0.0549334$
$R_{16}^* = 0.13832$	$\gamma_{16} = 1.2407467$	$F(R_{16}^*) = 0.676898$	$\epsilon_{16} = 0.0547652$
$R_{17}^* = 0.13903$	$\gamma_{17} = 1.2421809$	$F(R_{17}^*) = 0.676999$	$\epsilon_{17} = 0.0546119$
$R_{18}^* = 0.13968$	$\gamma_{18} = 1.2435107$	$F(R_{18}^*) = 0.677092$	$\epsilon_{18} = 0.0544711$
$R_{19}^* = 0.14029$	$\gamma_{19} = 1.2447186$	$F(R_{19}^*) = 0.677178$	$\epsilon_{19} = 0.0543411$
$R_{20}^* = 0.14085$	$\gamma_{20} = 1.2458665$	$F(R_{20}^*) = 0.677259$	$\epsilon_{20} = 0.0542204$
$R_{21}^* = 0.14138$	$\gamma_{21} = 1.2469242$	$F(R_{21}^*) = 0.677335$	$\epsilon_{21} = 0.0541076$
$R_{22}^* = 0.14187$	$\gamma_{22} = 1.2479438$	$F(R_{22}^*) = 0.677406$	$\epsilon_{22} = 0.0540017$
$R_{23}^* = 0.14234$	$\gamma_{23} = 1.2488869$	$F(R_{23}^*) = 0.677474$	$\epsilon_{23} = 0.0539018$
$R_{24}^* = 0.14278$	$\gamma_{24} = 1.2497992$	$F(R_{24}^*) = 0.677539$	$\epsilon_{24} = 0.0538072$
$R_{25}^* = 0.1432$	$\gamma_{25} = 1.2506663$	$F(R_{25}^*) = 0.6776$	$\epsilon_{25} = 0.0537172$
$R_{26}^* = 0.14361$	$\gamma_{26} = 1.2514715$	$F(R_{26}^*) = 0.677659$	$\epsilon_{26} = 0.0536313$
$R_{27}^* = 0.14399$	$\gamma_{27} = 1.2522842$	$F(R_{27}^*) = 0.677716$	$\epsilon_{27} = 0.0535489$
$R_{28}^* = 0.14437$	$\gamma_{28} = 1.2530263$	$F(R_{28}^*) = 0.67777$	$\epsilon_{28} = 0.0534697$
$R_{29}^* = 0.14473$	$\gamma_{29} = 1.2537652$	$F(R_{29}^*) = 0.677823$	$\epsilon_{29} = 0.0533932$
$R_{30}^* = 0.14508$	$\gamma_{30} = 1.2544791$	$F(R_{30}^*) = 0.677875$	$\epsilon_{30} = 0.0533192$
$R_{31}^* = 0.14542$	$\gamma_{31} = 1.255175$	$F(R_{31}^*) = 0.677925$	$\epsilon_{31} = 0.0532473$

$R^*_{32} = 0.14575$	$\gamma_{32} = 1.2558593$	$F(R^*_{32}) = 0.677974$	$\epsilon_{32} = 0.0531771$
$R^*_{33} = 0.14607$	$\gamma_{33} = 1.2565379$	$F(R^*_{33}) = 0.678022$	$\epsilon_{33} = 0.0531086$
$R^*_{34} = 0.14639$	$\gamma_{34} = 1.2571865$	$F(R^*_{34}) = 0.67807$	$\epsilon_{34} = 0.0530414$
$R^*_{35} = 0.1467$	$\gamma_{35} = 1.2578398$	$F(R^*_{35}) = 0.678116$	$\epsilon_{35} = 0.0529754$
$R^*_{36} = 0.14701$	$\gamma_{36} = 1.2584728$	$F(R^*_{36}) = 0.678162$	$\epsilon_{36} = 0.0529103$
$R^*_{37} = 0.14732$	$\gamma_{37} = 1.2590901$	$F(R^*_{37}) = 0.678208$	$\epsilon_{37} = 0.0528459$
$R^*_{38} = 0.14762$	$\gamma_{38} = 1.2597255$	$F(R^*_{38}) = 0.678253$	$\epsilon_{38} = 0.0527821$
$R^*_{39} = 0.14792$	$\gamma_{39} = 1.2603536$	$F(R^*_{39}) = 0.678299$	$\epsilon_{39} = 0.0527187$
$R^*_{40} = 0.14822$	$\gamma_{40} = 1.2609785$	$F(R^*_{40}) = 0.678344$	$\epsilon_{40} = 0.0526554$
$R^*_{41} = 0.14852$	$\gamma_{41} = 1.2616042$	$F(R^*_{41}) = 0.678389$	$\epsilon_{41} = 0.0525923$
$R^*_{42} = 0.14883$	$\gamma_{42} = 1.2622052$	$F(R^*_{42}) = 0.678435$	$\epsilon_{42} = 0.0525289$
$R^*_{43} = 0.14913$	$\gamma_{43} = 1.2628447$	$F(R^*_{43}) = 0.678481$	$\epsilon_{43} = 0.0524653$
$R^*_{44} = 0.14943$	$\gamma_{44} = 1.2634973$	$F(R^*_{44}) = 0.678527$	$\epsilon_{44} = 0.0524013$
$R^*_{45} = 0.14974$	$\gamma_{45} = 1.2641378$	$F(R^*_{45}) = 0.678574$	$\epsilon_{45} = 0.0523366$
$R^*_{46} = 0.15006$	$\gamma_{46} = 1.2647704$	$F(R^*_{46}) = 0.678622$	$\epsilon_{46} = 0.052271$
$R^*_{47} = 0.15038$	$\gamma_{47} = 1.2654297$	$F(R^*_{47}) = 0.67867$	$\epsilon_{47} = 0.0522044$
$R^*_{48} = 0.1507$	$\gamma_{48} = 1.2661208$	$F(R^*_{48}) = 0.67872$	$\epsilon_{48} = 0.0521367$
$R^*_{49} = 0.15103$	$\gamma_{49} = 1.2668193$	$F(R^*_{49}) = 0.678771$	$\epsilon_{49} = 0.0520674$
$R^*_{50} = 0.15138$	$\gamma_{50} = 1.2675011$	$F(R^*_{50}) = 0.678823$	$\epsilon_{50} = 0.0519965$
$R^*_{51} = 0.15173$	$\gamma_{51} = 1.2682323$	$F(R^*_{51}) = 0.678876$	$\epsilon_{51} = 0.0519237$
$R^*_{52} = 0.15209$	$\gamma_{52} = 1.26899$	$F(R^*_{52}) = 0.678932$	$\epsilon_{52} = 0.0518486$
$R^*_{53} = 0.15246$	$\gamma_{53} = 1.2697821$	$F(R^*_{53}) = 0.67899$	$\epsilon_{53} = 0.0517711$
$R^*_{54} = 0.15285$	$\gamma_{54} = 1.2705873$	$F(R^*_{54}) = 0.679049$	$\epsilon_{54} = 0.0516907$
$R^*_{55} = 0.15325$	$\gamma_{55} = 1.2714451$	$F(R^*_{55}) = 0.679112$	$\epsilon_{55} = 0.051607$
$R^*_{56} = 0.15368$	$\gamma_{56} = 1.2723064$	$F(R^*_{56}) = 0.679177$	$\epsilon_{56} = 0.0515197$
$R^*_{57} = 0.15412$	$\gamma_{57} = 1.273244$	$F(R^*_{57}) = 0.679246$	$\epsilon_{57} = 0.0514283$
$R^*_{58} = 0.15459$	$\gamma_{58} = 1.2742119$	$F(R^*_{58}) = 0.679319$	$\epsilon_{58} = 0.0513321$
$R^*_{59} = 0.15508$	$\gamma_{59} = 1.2752568$	$F(R^*_{59}) = 0.679396$	$\epsilon_{59} = 0.0512306$
$R^*_{60} = 0.1556$	$\gamma_{60} = 1.2763678$	$F(R^*_{60}) = 0.679478$	$\epsilon_{60} = 0.051123$
$R^*_{61} = 0.15616$	$\gamma_{61} = 1.2775371$	$F(R^*_{61}) = 0.679566$	$\epsilon_{61} = 0.0510085$
$R^*_{62} = 0.15676$	$\gamma_{62} = 1.2787913$	$F(R^*_{62}) = 0.67966$	$\epsilon_{62} = 0.0508861$
$R^*_{63} = 0.1574$	$\gamma_{63} = 1.280162$	$F(R^*_{63}) = 0.679762$	$\epsilon_{63} = 0.0507544$
$R^*_{64} = 0.1581$	$\gamma_{64} = 1.2816267$	$F(R^*_{64}) = 0.679872$	$\epsilon_{64} = 0.0506122$
$R^*_{65} = 0.15886$	$\gamma_{65} = 1.2832312$	$F(R^*_{65}) = 0.679993$	$\epsilon_{65} = 0.0504577$

$R^*_{66} = 0.15969$	$\gamma_{66} = 1.2850016$	$F(R^*_{66}) = 0.680126$	$\epsilon_{66} = 0.0502888$
$R^*_{67} = 0.16061$	$\gamma_{67} = 1.2869469$	$F(R^*_{67}) = 0.680274$	$\epsilon_{67} = 0.0501029$
$R^*_{68} = 0.16163$	$\gamma_{68} = 1.2891241$	$F(R^*_{68}) = 0.680439$	$\epsilon_{68} = 0.0498968$
$R^*_{69} = 0.16277$	$\gamma_{69} = 1.2915835$	$F(R^*_{69}) = 0.680625$	$\epsilon_{69} = 0.0496664$
$R^*_{70} = 0.16407$	$\gamma_{70} = 1.2943462$	$F(R^*_{70}) = 0.680837$	$\epsilon_{70} = 0.0494067$
$R^*_{71} = 0.16555$	$\gamma_{71} = 1.2975399$	$F(R^*_{71}) = 0.681081$	$\epsilon_{71} = 0.0491107$
$R^*_{72} = 0.16727$	$\gamma_{72} = 1.3012328$	$F(R^*_{72}) = 0.681366$	$\epsilon_{72} = 0.0487696$
$R^*_{73} = 0.16928$	$\gamma_{73} = 1.305618$	$F(R^*_{73}) = 0.681704$	$\epsilon_{73} = 0.0483715$
$R^*_{74} = 0.17169$	$\gamma_{74} = 1.310843$	$F(R^*_{74}) = 0.682112$	$\epsilon_{74} = 0.0478995$
$R^*_{75} = 0.17461$	$\gamma_{75} = 1.3172771$	$F(R^*_{75}) = 0.682614$	$\epsilon_{75} = 0.0473301$
$R^*_{76} = 0.17825$	$\gamma_{76} = 1.3253364$	$F(R^*_{76}) = 0.68325$	$\epsilon_{76} = 0.0466283$
$R^*_{77} = 0.18292$	$\gamma_{77} = 1.3357437$	$F(R^*_{77}) = 0.68408$	$\epsilon_{77} = 0.0457406$
$R^*_{78} = 0.18912$	$\gamma_{78} = 1.3497665$	$F(R^*_{78}) = 0.68521$	$\epsilon_{78} = 0.0445811$
$R^*_{79} = 0.19775$	$\gamma_{79} = 1.3696989$	$F(R^*_{79}) = 0.686838$	$\epsilon_{79} = 0.0430022$
$R^*_{80} = 0.21061$	$\gamma_{80} = 1.4001298$	$F(R^*_{80}) = 0.689369$	$\epsilon_{80} = 0.0407297$
$R^*_{81} = 0.23172$	$\gamma_{81} = 1.4523369$	$F(R^*_{81}) = 0.693804$	$\epsilon_{81} = 0.0371936$
$R^*_{82} = 0.27259$	$\gamma_{82} = 1.5620635$	$F(R^*_{82}) = 0.703325$	$\epsilon_{82} = 0.0309996$
$R^*_{83} = 0.32069$	$\gamma_{83} = 1.7081291$	$F(R^*_{83}) = 0.716$	$\epsilon_{83} = 0.0177022$

Chapter 5

Conclusion

We conclude by discussing some directions for future work for the problems considered in this thesis and some of their variants.

5.1 Future Directions - The RMFC Problem

Regarding this problem the main important question is to whether it is possible to find an asymptotic polynomial time approximation scheme (**APTAS**) or not. We believe that it should be possible to have an asymptotic **PTAS** for the RMFC problem. Perhaps one way is to somehow guess the upper part of the optimal solution in polynomial time and then use the LP to round the solution for the height reduced instance for which we initially applied the height reduction lemma.

Recall that the RMFC problem on trees does not admit better than 2-approximation unless **NP** = **NP** [57]. However, this does not rule out the possibility of a +1 approximation or an asymptotic **PTAS**. Our result is an indication that it is plausible that an asymptotic **PTAS** exists, especially since the exponent is $O(\log\log n)$, not $O(\log n)$.

5.2 Future Directions - Throughput Maximization

There are so many possible directions regarding the Throughput Maximization problem. Here are a summary of the questions that seem interesting but we could not answer:

- **(Q)PTAS for general c and $m = O(1)$:**

  One interesting and important question is if it is possible to find a **PTAS** using our approach (cutting heads and tails and using DP) for general c. Currently the best algorithm for unweighted throughput maximization on $m = O(1)$ machines for general c has approximation ratio $(\alpha_0 - \epsilon)$ (running in time $n^{O(m/\epsilon^5)}$) for some absolute $\alpha_0 > 1 - 1/e$ [49]. So finding a **PTAS** (or even a **QPTAS**) would be a huge improvement.

  Recall that there is another algorithm with ratio $1 - O(\sqrt{(\log m)/m} - \epsilon)$ (for any $\epsilon > 0$) on m machines [49]. This ratio approaches 1 as m grows, so, the only case we should try to improve is when the number of machines is constant.

  Here is what we think might be a good approach towards such result. Based on Lemma 8 we can remove the heads and tails of the span of all the jobs with relatively small processing times (with respect to their span size) to have a more structured instance. But this is coming with a loss of $O(\epsilon c)$ fraction of the total number of jobs where c is the number of distinct processing times.

  Improving Lemma 8 by considering all the jobs at the same time (instead of once for each processing time $p \in P$) to reduce the loss to only an $O(\epsilon)$ fraction of the jobs, would eliminate the need to introduce $\varepsilon' = \varepsilon/c$, so the total running time of the algorithm would be much better.

  Also, reducing the running time of the dynamic programming that we use to prove Theorem 11 to $\varepsilon^{-3}(\log n/\epsilon)^{O(\varepsilon^{-6}c)} \log T$ (right now the running time of the dynamic programming part of our algorithm is $\varepsilon^{-3} n^{O(\varepsilon^{-6}c)} \log T$) with possibly another $O(\epsilon)$ loss in the objective function using a rounding technique, will give a **PTAS** for a more general case of $c = O(\frac{\log n}{\log \log n})$.

- **NP-Hardness and APX-Hardness Results**:

  There are only two main hardness results for Throughput Maximization Problem. Spieksma [72] showed that the discrete version of the problem is $MAX\text{-}SNP$ hard using a reduction to a version of $MAX\text{-}3SAT$. For the continuous version, it has been shown in [35] that even for $m = 1$ and $p_j \in \{p, q\}$ where p and q are strictly greater than 1 the problem is **NP**-Complete. So, there could be so many different interesting questions regarding the **NP**-hardness or **APX**-hardness of Throughput Maximization Problem and its different special cases. One main question that seems more interesting is to check if the problem is **APX**-Hard for general c or not.

- **Throughput Maximization with Resource Augmentation**:

  As an easy corollary, the **PTAS** we present for c distinct processing times implies a bi-criteria **QPTAS** as well, i.e. a $(1 - \epsilon)$-approximation using $(1 + \epsilon)$-speed up with quasi-polynomial running time for general values of c. This would be the same as the best known algorithm for the problem presented by [50]. One interesting question is to find a bi-criteria **PTAS** for throughput maximization with resource augmentation. An obvious way to achieve this is to improve the running time of our dynamic programming to be polynomial when $c = polylog(n)$. Another possible way is to use the machine speed-up, not only for rounding the processing times to reduce c to $\log n$, but also throughout the algorithm. Adjusting Lemma 8 considering the speed-up, to reduce the loss from $O(\epsilon c)$ to $O(\epsilon)$, and changing our dynamic programming approach to utilize the speed-up ability and reduce the running time even more, would result in a **PTAS** for a more general case of $c = O(\frac{\log n}{\log \log n})$.

- **Machine Minimization Problem**:

  Another interesting open question is whether it is possible to use our method to improve the best ratios in the Machine Minimization problem or not. For the problem of machine minimization, where we have to find the minimum number of machines with which we can schedule all the jobs, the algorithm provided in [66] has approximation ratio $O(\sqrt{\log n / \log \log n})$ only when $OPT = \Omega(\sqrt{\log n / \log \log n})$, and ratio $O(1)$ when $OPT = \Omega(\log n)$. Later Chuzhoy et al. [27] presented an $O(OPT)$-approximation which is good for the instances with relatively small OPT. Combining this with the earlier works implies an $O(\sqrt{\log n / \log\log n})$-approximation. Chuzhoy and Naor [28] showed a hardness of $\Omega(\log\log n)$ for the machine minimization problem.

- **Resource Allocation Extension**:

  Another generalization of the problem is when we assign a height to each job as well and allow them to share the machine as long as the total height of all the jobs running on a machine at the same time is no more than 1. The first approximation algorithm for this generalization is provided by Bar-Noy et al. [13] which has ratio 1/5. Chuzhoy et al. [29] improved it by providing an $(e-1)/(2e-1) > 0.3873$-approximation algorithm which is only working for the unweighted and discrete version of the problem. So, another direction is to check if our approach could improve the approximation ratio for resource allocation problem or not.

  There is also another extension of the problem where each job j needs m_j many machines at the same time to be able to be processed. In the regular Throughput Maximization $m_j = 1$ for all jobs. This could also be a good direction to check if our approach could be extended.

5.3 Future Directions - Hierarchical Clustering

This is a list of the open questions one can consider regarding this problem:

- **Providing Algorithms with Better Approximation Ratios for $Rev_{Dual}(T)$**:

  As we discussed in the previous work section Alon et al. [5] proved that the algorithm of [3] is actually giving a $2\rho/3 = 0.585$-approximation, by detecting the cases where average-linkage is not good and provided another algorithm which performs better in those cases, using Maximum Un-Cut Bisection in the first step and Average Linkage (or Random) for the remaining clusters. Then they managed to show that, by taking the better of the two solutions the approximation ratio would be better.

One question is to see whether using a similar "peel off" phase at the beginning and then choosing the better of the two of Random (or Average Linkage) and Minimum Cut (Maximum Un-Cut) would make any improvement in the approximation ratio, or not.

When we have $w_{i,j}$s as similarities, then it makes sense to peel off those nodes with $W_v < \gamma\frac{2W}{n}$, or those with less than average similarity to others. Then, what remains is a set of nodes with big similarities to each others. Running the better of the two of Minimum Cut and Random might lead to a better approximation ratio.

- **Hardness Results for** $Cost_H(T)$:

  As we discussed before, there is an **NP**-hardness result for the problem of minimizing $Cost(T)$ [32] and **APX**-Hardness result for the problem of maximizing $Rev(T)$ [3]. We believe that it could be possible to find such results for the newly introduced objective function $Cost_H(T)$.

References

[1] D. Adjiashvili, A. Baggio, and R. Zenklusen, "Firefighting on trees beyond integrality gaps," *ACM Trans. Algorithms*, vol. 15, no. 2, 20:1–20:33, 2019. DOI: 10.1145/3173046. [Online]. Available: https://doi.org/10.1145/3173046. 9, 10, 16–21, 25, 30–34

[2] M. Adler, A. L. Rosenberg, R. K. Sitaraman, and W. Unger, "Scheduling time-constrained communication in linear networks," *Theory Comput. Syst.*, vol. 35, no. 6, pp. 599–623, 2002. [Online]. Available: https://doi.org/10.1007/s00224-002-1001-6. 11

[3] S. Ahmadian, V. Chatziafratis, A. Epasto, *et al.*, "Bisect and conquer: Hierarchical clustering via max-uncut bisection," in *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 3-5 June 2020, Palermo, Sicily, Italy*, 2020. 73, 74, 84, 98, 99

[4] S. Ahmadian, A. Epasto, M. Knittel, *et al.*, "Fair hierarchical clustering," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin, Eds., 2020. 75

[5] N. Alon, Y. Azar, and D. Vainstein, "Hierarchical clustering: A 0.585 revenue approximation," in *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*, J. D. Abernethy and S. Agarwal, Eds., ser. Proceedings of Machine Learning Research, vol. 125, PMLR, 2020, pp. 153–162. [Online]. Available: http://proceedings.mlr.press/v125/alon20b.html. 74, 98

[6] E. Anshelevich, D. Chakrabarty, A. Hate, and C. Swamy, "Approximability of the firefighter problem - computing cuts over time," *Algorithmica*, vol. 62, no. 1-2, pp. 520–536, 2012. 17

[7] S. Arora, S. Rao, and U. V. Vazirani, "Expander flows, geometric embeddings and graph partitioning," *J. ACM*, vol. 56, no. 2, 5:1–5:37, 2009. DOI: 10.1145/1502793.1502794. [Online]. Available: https://doi.org/10.1145/1502793.1502794. 69

[8] P. Austrin, S. Benabbas, and K. Georgiou, “Better balance by being biased: A 0.8776-approximation for max bisection,” *ACM Trans. Algorithms*, vol. 13, no. 1, 2:1–2:27, 2016. [Online]. Available: `https://doi.org/10.1145/2907052`. 74

[9] D. Bakkelund, “Order preserving hierarchical agglomerative clustering of strict posets,” *CoRR*, vol. abs/2004.12488, 2020. arXiv: `2004.12488`. [Online]. Available: `https://arxiv.org/abs/2004.12488`. 74

[10] N. Bansal, H.-L. Chan, R. Khandekar, K. Pruhs, C. Stein, and B. Schieber, “Non-preemptive min-sum scheduling with resource augmentation,” in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, 2007, pp. 614–624. [Online]. Available: `https://doi.org/10.1109/FOCS.2007.46`. 38

[11] P. Baptiste, “On minimizing the weighted number of late jobs in unit execution time open-shops,” *European Journal of Operational Research*, vol. 149, no. 2, pp. 344–354, 2003. [Online]. Available: `https://doi.org/10.1016/S0377-2217(02)00759-2`. 37

[12] P. Baptiste, P. Brucker, S. Knust, and V. G. Timkovsky, “Ten notes on equal-processing-time scheduling,” *4OR*, vol. 2, no. 2, pp. 111–127, 2004. [Online]. Available: `https://doi.org/10.1007/s10288-003-0024-4`. 11, 37

[13] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber, “A unified approach to approximating resource allocation and scheduling,” *J. ACM*, vol. 48, no. 5, pp. 1069–1090, 2001. [Online]. Available: `https://doi.org/10.1145/502102.502107`. 37, 38, 98

[14] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, “Approximating the throughput of multiple machines in real-time scheduling,” *SIAM J. Comput.*, vol. 31, no. 2, pp. 331–352, 2001. [Online]. Available: `https://doi.org/10.1137/S0097539799354138`. 36

[15] S. K. Baruah, G. Koren, D. Mao, *et al.*, “On the competitiveness of on-line real-time task scheduling,” *Real-Time Systems*, vol. 4, no. 2, pp. 125–144, 1992. [Online]. Available: `https://doi.org/10.1007/BF00365406`. 39

[16] P. Berman and B. DasGupta, “Improvements in throughout maximization for real-time scheduling,” in *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, 2000, pp. 680–687. [Online]. Available: `https://doi.org/10.1145/335305.335401`. 37

[17] L. Cai, E. Verbin, and L. Yang, “Firefighting on trees: (1-1/e)-approximation, fixed parameter tractability and a subexponential algorithm,” in *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5369, Springer, 2008, pp. 258–269. 17, 18

[18] G. Calinescu, C. Chekuri, M. Pal, and J. Vondrak, "Maximizing a monotone submodular function subject to a matroid constraint," *SIAM J. Comput.*, vol. 40, no. 6, pp. 1740–1766, 2011. 18

[19] P. Chalermsook and J. Chuzhoy, "Resource minimization for fire containment," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, SIAM, 2010, pp. 1334–1349. 9, 17, 19

[20] P. Chalermsook and D. Vaz, "New integrality gap results for the firefighters problem on trees," in *Approximation and Online Algorithms - 14th International Workshop, WAOA 2016, Aarhus, Denmark, August 25-26, 2016, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 10138, Springer, 2016, pp. 65–77. 18

[21] M. Charikar and V. Chatziafratis, "Approximate hierarchical clustering via sparsest cut and spreading metrics," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, 2017, pp. 841–854. [Online]. Available: `https://doi.org/10.1137/1.9781611974782.53`. 14, 69–71

[22] M. Charikar, V. Chatziafratis, and R. Niazadeh, "Hierarchical clustering better than average-linkage," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, SIAM, 2019, pp. 2291–2304. [Online]. Available: `https://doi.org/10.1137/1.9781611975482.139`. 14, 66, 73–77, 79, 82

[23] M. Charikar, V. Chatziafratis, R. Niazadeh, and G. Yaroslavtsev, "Hierarchical clustering for euclidean data," in *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, K. Chaudhuri and M. Sugiyama, Eds., ser. Proceedings of Machine Learning Research, vol. 89, PMLR, 2019, pp. 2721–2730. [Online]. Available: `http://proceedings.mlr.press/v89/charikar19a.html`. 74

[24] V. Chatziafratis, N. Gupta, and E. Lee, "Inapproximability for local correlation clustering and dissimilarity hierarchical clustering," *CoRR*, vol. abs/2010.01459, 2020. arXiv: `2010.01459`. [Online]. Available: `https://arxiv.org/abs/2010.01459`. 14

[25] V. Chatziafratis, R. Niazadeh, and M. Charikar, "Hierarchical clustering with structural constraints," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, J. G. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 773–782. [Online]. Available: `http://proceedings.mlr.press/v80/chatziafratis18a.html`. 74, 79, 80

[26] M. H. Chehreghani, "Hierarchical correlation clustering and tree preserving embedding," *CoRR*, vol. abs/2002.07756, 2020. arXiv: 2002.07756. [Online]. Available: https://arxiv.org/abs/2002.07756. 75

[27] J. Chuzhoy, S. Guha, S. Khanna, and J. Naor, "Machine minimization for scheduling jobs with interval constraints," in *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, 2004, pp. 81–90. [Online]. Available: https://doi.org/10.1109/FOCS.2004.38. 38, 98

[28] J. Chuzhoy and J. Naor, "New hardness results for congestion minimization and machine scheduling," *J. ACM*, vol. 53, no. 5, pp. 707–721, 2006. [Online]. Available: https://doi.org/10.1145/1183907.1183908. 38, 98

[29] J. Chuzhoy, R. Ostrovsky, and Y. Rabani, "Approximation algorithms for the job interval selection problem and related scheduling problems," *Math. Oper. Res.*, vol. 31, no. 4, pp. 730–738, 2006. [Online]. Available: https://doi.org/10.1287/moor.1060.0218. 37, 38, 44, 98

[30] V. Cohen-Addad, V. Kanade, and F. Mallmann-Trenn, "Hierarchical clustering beyond the worst-case," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 6201–6209. [Online]. Available: http://papers.nips.cc/paper/7200-hierarchical-clustering-beyond-the-worst-case. 72

[31] V. Cohen-Addad, V. Kanade, F. Mallmann-Trenn, and C. Mathieu, "Hierarchical clustering: Objective functions and algorithms," *J. ACM*, vol. 66, no. 4, 26:1–26:42, 2019. [Online]. Available: https://doi.org/10.1145/3321386. 13, 66

[32] S. Dasgupta, "A cost function for similarity-based hierarchical clustering," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, 2016, pp. 118–127. [Online]. Available: https://doi.org/10.1145/2897518.2897527. 13, 14, 66, 67, 69, 71, 99

[33] L. Dhulipala, D. Eisenstat, J. Lacki, V. S. Mirrokni, and J. Shi, "Hierarchical agglomerative graph clustering in nearly-linear time," *CoRR*, vol. abs/2106.05610, 2021. arXiv: 2106.05610. [Online]. Available: https://arxiv.org/abs/2106.05610. 75

[34] M. Dourado, R. Rodrigues, and J. Szwarcfiter, "Scheduling unit time jobs with integer release dates to minimize the weighted number of tardy jobs," *Annals of Operations Research*, vol. 169, no. 1, pp. 81–91, 2009. [Online]. Available: https://EconPapers.repec.org/RePEc:spr:annopr:v:169:y:2009:i:1:p:81-91:10.1007/s10479-008-0479-y. 11, 37

[35] J. Elffers and M. de Weerdt, "Scheduling with two non-unit task lengths is np-complete," *CoRR*, vol. abs/1412.3095, 2014. arXiv: 1412.3095. [Online]. Available: http://arxiv.org/abs/1412.3095. 11, 97

[36] E. Emamjomeh-Zadeh and D. Kempe, “Adaptive hierarchical clustering using ordinal queries,” in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, A. Czumaj, Ed., SIAM, 2018, pp. 415–429. DOI: 10.1137/1.9781611975031.28. [Online]. Available: https://doi.org/10.1137/1.9781611975031.28. 74

[37] U. Feige, L. Lovász, and P. Tetali, “Approximating min sum set cover,” *Algorithmica*, vol. 40, no. 4, pp. 219–234, 2004. [Online]. Available: https://doi.org/10.1007/s00453-004-1110-5. 89

[38] S. Finbow, A. D. King, G. MacGillivray, and R. Rizzi, “The firefighter problem for graphs of maximum degree three,” *Discret. Math.*, vol. 307, no. 16, pp. 2094–2105, 2007. 17

[39] M. Fischetti, S. Martello, and P. Toth, “The fixed job schedule problem with working-time constraints,” *Operations Research*, vol. 37, no. 3, pp. 395–403, 1989. [Online]. Available: https://doi.org/10.1287/opre.37.3.395. 11

[40] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979, ISBN: 0-7167-1044-7. 11

[41] M. X. Goemans and D. P. Williamson, “Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming,” *J. ACM*, vol. 42, no. 6, pp. 1115–1145, 1995. [Online]. Available: https://doi.org/10.1145/227683.227684. 15, 67, 78, 80, 81

[42] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. North-Holland Publishing Co. Amsterdam, The Netherlands, 2004, ISBN: 0-4445-1530-5. 44

[43] R. H. M. Hafez and G. R. Rajugopal, “Adaptive rate controlled, robust video communication over packet wireless networks,” *MONET*, vol. 3, no. 1, pp. 33–47, 1998. [Online]. Available: https://doi.org/10.1023/A:1019156211458. 11

[44] B. Hartnell and Q. Li, “Firefighting on trees: How bad is the greedy algorithm?” In *Proceedings of Congressus Numerantium*, 2000, pp. 187–192. 17

[45] B. Hartnell, “Firefighter! an application of domination,” in *24th Manitoba Conference on Combinatorial Mathematics and Computing*, University of Manitoba in Winnipeg, Canada, 1995. 9

[46] S. Høgemo, C. Paul, and J. A. Telle, “Hierarchical clusterings of unweighted graphs,” in *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, J. Esparza and D. Král’, Eds., ser. LIPIcs, vol. 170, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 47:1–

47:13. DOI: 10.4230/LIPIcs.MFCS.2020.47. [Online]. Available: https://doi.org/10.4230/LIPIcs.MFCS.2020.47. 75

[47] D. Hyatt-Denesik, "Algorithms in throughput maximization," M.S. thesis, University of Alberta, 2019. [Online]. Available: https://doi.org/10.7939/r3-z2c9-qq71. 39

[48] D. Hyatt-Denesik, M. Rahgoshay, and M. R. Salavatipour, "Approximations for throughput maximization," in *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, ser. LIPIcs, vol. 181, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 11:1–11:17. DOI: 10.4230/LIPIcs.ISAAC.2020.11. [Online]. Available: https://doi.org/10.4230/LIPIcs.ISAAC.2020.11. v

[49] S. Im, S. Li, and B. Moseley, "Breaking 1 - 1/e barrier for non-preemptive throughput maximization," in *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, 2017, pp. 292–304. [Online]. Available: https://doi.org/10.1007/978-3-319-59250-3%5C_24. 37, 96

[50] S. Im, S. Li, B. Moseley, and E. Torng, "A dynamic programming framework for non-preemptive scheduling problems on multiple machines [extended abstract]," in *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, P. Indyk, Ed., SIAM, 2015, pp. 1070–1086. [Online]. Available: https://doi.org/10.1137/1.9781611973730.72. 12, 38, 54, 97

[51] Y. Iwaikawa, N. Kamiyama, and T. Matsui, "Improved approximation algorithms for firefighter problem on trees," *IEICE Trans. Inf. Syst.*, vol. 94-D, no. 2, pp. 196–199, 2011. 18

[52] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010. 12

[53] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Comb.*, vol. 4, no. 4, pp. 373–396, 1984. DOI: 10.1007/BF02579150. [Online]. Available: https://doi.org/10.1007/BF02579150. 8

[54] S. Khot, "On the power of unique 2-prover 1-round games," in *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, J. H. Reif, Ed., ACM, 2002, pp. 767–775. DOI: 10.1145/509907.510017. [Online]. Available: https://doi.org/10.1145/509907.510017. 6

[55] S. Khot, D. Minzer, and M. Safra, "On independent sets, 2-to-2 games, and grassmann graphs," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, ACM, 2017, pp. 576–589. DOI: 10.1145/3055399.3055432. [Online]. Available: https://doi.org/10.1145/3055399.3055432. 7

[56] ——, "Pseudorandom sets in grassmann graph have near-perfect expansion," in *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, IEEE Computer Society, 2018, pp. 592–601. DOI: `10.1109/FOCS.2018.00062`. [Online]. Available: `https://doi.org/10.1109/FOCS.2018.00062`. 7

[57] A. D. King and G. MacGillivray, "The firefighter problem for cubic graphs," *Discret. Math.*, vol. 310, no. 3, pp. 614–621, 2010. 10, 17, 95

[58] E. Lee, "Improved hardness for cut, interdiction, and firefighter problems," in *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, ser. LIPIcs, vol. 80, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 92:1–92:14. 18

[59] H. Liu and M. E. Zarki, "Adaptive source rate control for real-time wireless video transmission," *MONET*, vol. 3, no. 1, pp. 49–60, 1998. [Online]. Available: `https://doi.org/10.1023/A:1019108328296`. 11

[60] F. McSherry, "Spectral partitioning of random graphs," in *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, IEEE Computer Society, 2001, pp. 529–537. [Online]. Available: `https://doi.org/10.1109/SFCS.2001.959929`. 73

[61] A. K. Menon, A. Rajagopalan, B. Sumengen, G. Citovsky, Q. Cao, and S. Kumar, "Online hierarchical clustering approximations," *CoRR*, vol. abs/1909.09667, 2019. arXiv: `1909.09667`. [Online]. Available: `http://arxiv.org/abs/1909.09667`. 75

[62] B. Moseley, S. Vassilvitskii, and Y. Wang, "Hierarchical clustering in general metric spaces using approximate nearest neighbors," in *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*, A. Banerjee and K. Fukumizu, Eds., ser. Proceedings of Machine Learning Research, vol. 130, PMLR, 2021, pp. 2440–2448. [Online]. Available: `http://proceedings.mlr.press/v130/moseley21a.html`. 75

[63] B. Moseley and J. Wang, "Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 3094–3103. [Online]. Available: `http://papers.nips.cc/paper/6902-approximation-bounds-for-hierarchical-clustering-average-linkage-bisecting-k-means-and-local-search`. 73

[64] C. N. Potts and V. A. Strusevich, "Fifty years of scheduling: A survey of milestones," *JORS*, vol. 60, no. S1, 2009. [Online]. Available: `https://doi.org/10.1057/jors.2009.2`. 10

[65] K. Pruhs, J. Sgall, and E. Torng, “Online scheduling,” in *Handbook of Scheduling - Algorithms, Models, and Performance Analysis.* 2004. [Online]. Available: `http://www.crcnetbase.com/doi/abs/10.1201/9780203489802.ch15`. 38

[66] P. Raghavan and C. D. Thompson, “Randomized rounding: A technique for provably good algorithms and algorithmic proofs,” *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987. [Online]. Available: `https://doi.org/10.1007/BF02579324`. 38, 97

[67] P. Raghavendra and D. Steurer, “Graph expansion and the unique games conjecture,” in *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, 2010, pp. 755–764. [Online]. Available: `https://doi.org/10.1145/1806689.1806792`. 70, 74

[68] M. Rahgoshay and M. R. Salavatipour, “Asymptotic quasi-polynomial time approximation scheme for resource minimization for fire containment,” in *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, ser. LIPIcs, vol. 154, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 33:1–33:14. DOI: `10.4230/LIPIcs.STACS.2020.33`. [Online]. Available: `https://doi.org/10.4230/LIPIcs.STACS.2020.33`. v

[69] A. Roy and S. Pokutta, “Hierarchical clustering via spreading metrics,” ser. CEUR Workshop Proceedings, vol. 6325, CEUR-WS.org, 2016. [Online]. Available: `http://papers.nips.cc/paper/6325-hierarchical-clustering-via-spreading-metrics.pdf`. 69, 70

[70] P. Schuurman and G. J. Woeginger, “Polynomial time approximation algorithms for machine scheduling: Ten open problems,” *Journal of Scheduling*, vol. 2, no. 5, pp. 203–213, 1999. 10

[71] J. Sgall, “Open problems in throughput scheduling,” in *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, 2012, pp. 2–11. [Online]. Available: `https://doi.org/10.1007/978-3-642-33090-2%5C_2`. 10, 11

[72] F. C. R. Spieksma, “Approximating an interval scheduling problem,” in *Approximation Algorithms for Combinatorial Optimization, International Workshop APPROX'98, Aalborg, Denmark, July 18-19, 1998, Proceedings*, 1998, pp. 169–180. [Online]. Available: `https://doi.org/10.1007/BFb0053973`. 36, 37, 96

[73] M. D. Summa, D. Pritchard, and L. Sanità, “Finding the closest ultrametric,” *Discrete Applied Mathematics*, vol. 180, pp. 70–80, 2015. [Online]. Available: `https://doi.org/10.1016/j.dam.2014.07.023`. 70

[74] D. Vainstein, V. Chatziafratis, G. Citovsky, A. Rajagopalan, M. Mahdian, and Y. Azar, “Hierarchical clustering via sketches and hierarchical correlation clustering,” in *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*, A. Banerjee and K. Fukumizu, Eds., ser. Proceedings of Machine Learning Research, vol. 130, PMLR, 2021, pp. 559–567. [Online]. Available: `http://proceedings.mlr.press/v130/vainstein21a.html`. 75

[75] V. V. Vazirani, *Approximation algorithms.* Springer, 2001, ISBN: 978-3-540-65367-7. [Online]. Available: `http://www.springer.com/computer/theoretical+computer+science/book/978-3-540-65367-7`. 2, 5

[76] D. Wang and Y. Wang, “An improved cost function for hierarchical cluster trees,” *J. Comput. Geom.*, vol. 11, no. 1, pp. 283–331, 2020. [Online]. Available: `https://journals.carleton.ca/jocg/index.php/jocg/article/view/431`. 74

[77] Y. Wang and B. Moseley, “An objective for hierarchical clustering in euclidean space and its connection to bisecting k-means,” in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, AAAI Press, 2020, pp. 6307–6314. [Online]. Available: `https://aaai.org/ojs/index.php/AAAI/article/view/6099`. 75

[78] D. B. West, *Introduction to Graph Theory*, 2nd ed. Prentice Hall, 2000, ISBN: 0130144002. 2

[79] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms.* Cambridge University Press, 2011, ISBN: 978-0-521-19527-0. [Online]. Available: `http://www.cambridge.org/de/knowledge/isbn/item5759340/`. 2

[80] D. K. Y. Yau and S. S. Lam, “Adaptive rate-controlled scheduling for multimedia applications,” *IEEE/ACM Trans. Netw.*, vol. 5, no. 4, pp. 475–488, 1997. [Online]. Available: `https://doi.org/10.1109/90.649461`. 11

[81] D. Zuckerman, “Linear degree extractors and the inapproximability of max clique and chromatic number,” *Theory Comput.*, vol. 3, no. 1, pp. 103–128, 2007. DOI: `10.4086/toc.2007.v003a006`. [Online]. Available: `https://doi.org/10.4086/toc.2007.v003a006`. 5