THE UNIVERSITY OF CHICAGO

APPROXIMATION ALGORITHMS FOR CAPACITATED K-MEDIAN AND SCHEDULING WITH RESOURCE AND PRECEDENCE CONSTRAINTS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
HÜSEYİN GÖKALP DEMİRCİ

CHICAGO, ILLINOIS
JUNE 2019

To my parents and my dear sister

TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES vii

ACKNOWLEDGMENTS viii

ABSTRACT ix

1 INTRODUCTION 1

2 CAPACITATED K-MEDIAN 3
- 2.1 Introduction 3
  - 2.1.1 Our Result 5
- 2.2 The Basic LP and the Configuration LP 7
- 2.3 Representatives, Black Components, and Groups 11
  - 2.3.1 Representatives, Bundles, and Initial Moving of Demands 11
  - 2.3.2 Black Components 13
  - 2.3.3 Groups 17
- 2.4 Constructing Local Solutions 19
  - 2.4.1 Concentrated Black Components and Earth Mover Distance 20
  - 2.4.2 Distributions of Local Solutions for Concentrated Components 22
  - 2.4.3 Local Solutions for Unions of Non-Concentrated Components 31
- 2.5 Rounding Algorithm 32
  - 2.5.1 Constructing Initial Set S^* of Open Facilities 34
  - 2.5.2 The remove procedure 37
  - 2.5.3 Obtaining the Final Solution 39

3 SCHEDULING WITH RESOURCE AND PRECEDENCE CONSTRAINTS . . . 42
- 3.1 Introduction 42
  - 3.1.1 Problem Definition 43
- 3.2 Related Work 44
- 3.3 Summary of Our Results and Techniques 49
  - 3.3.1 Theoretical Results 49
  - 3.3.2 Empirical Results 50
- 3.4 Identical Parallel Machines 51
  - 3.4.1 The Minimum Makespan Objective 51
  - 3.4.2 The Minimum Weighted Completion Time Objective 58
- 3.5 Uniformly Related Machines 62
- 3.6 An Empirical Study of the Divide-and-Conquer Algorithm 69
  - 3.6.1 Divide-and-Conquer vs Greedy Strategies 71
  - 3.6.2 Implementation of the Divide-and-Conquer Algorithm 73
  - 3.6.3 Experimental Setup 74
  - 3.6.4 Results 76

REFERENCES 82

LIST OF FIGURES

2.1 The three-phase clustering procedure. 11
2.2 The forest $\Upsilon^*_{\mathcal{J}}$ over $\mathcal{J}$ and $\mathcal{V}^{\mathsf{N}}$ 33

3.1 The intermediate schedule (right) obtained in the first step from the DAG (left) representing the precedence constraints prec. The intermediate schedule satisfies the precedences, but not necessarily the power constraints. 53
3.2 Percent overhead to theoretical lower bound on DAGs k-means, backprop, and npb-is. **DS** has significantly smaller overhead over all three greedy algorithms. . 76

LIST OF TABLES

3.1 Description of DAGs with node count n and height h 75
3.2 Percent improvement of **DS** over the best of **G1**, **G2**, and **G3** in each case with 10 and 20 machines and various power caps. 77
3.3 Percent overhead of **G1**|**G2**|**G3**|**DS**, respectively, on the theoretical lower bound. Lower is better. For each entry the best result is in bold and the second best is italic. 78
3.4 The algorithm scales well to 50 and 100 machines. Percent overhead of **G1**|**G2**|**G3**|**DS**, respectively, on the theoretical lower bound. 80

ACKNOWLEDGMENTS

First, I would like to thank my advisors, Janos Simon and Hank Hoffmann. Janos has essentially been my mentor and provided me with invaluable guidance in both academic and daily life over the years. I have gained deep insight into many subjects in academic research and Computer Science through my conversations with him. I was a student in Hank's class in my first year, but it took me a couple more years to approach him for potential research directions. He suggested studying a particular scheduling problem from both theory and systems points of view. Even though his research is primarily in systems, he was always on top of the progress we have made on the theory side. He provided me with key insights on how to combine theory and systems research and how to use techniques of one for solving problems in the other. He also has been one of the most supportive people in the department for me.

I am grateful to Shi Li for introducing me to one of the problems studied in this thesis and leading our collaboration. At the end of our very first meeting, he was able to suggest an open problem that a new PhD student would be able to make progress on with some hard work. This resulted in a collaboration on a paper together later on. I have learned how to use many advanced technical methods during our collaboration.

I would like to thank my colleagues and friends at the department and at TTIC, especially David Kim, Liwen Zhang, and Cafer Caferov for many tea breaks together with endless conversations. Also, I am in debt to all the friends I have made through the years in Chicago for allowing me to diversify my conversations away from Computer Science from time to time, in particular to Handan Acar. I am especially lucky to have met Katharina Maisel and I cannot thank her enough for all the support in the later years of my PhD.

Last and most important, I would like to thank my beloved family. Their hard work, unconditional love, and support made it possible for me to focus on getting a quality education and study what I enjoyed.

ABSTRACT

This thesis proposes approximation algorithms for two combinatorial optimization problems: 1) Capacitated k-Median, and 2) Scheduling with Resource and Precedence Constraints.

In the first problem, we are given a set of clients and a set of facilities in a metric space such that a set of k facilities need to be opened and each client needs to be assigned to an open facility (clustered around) to minimize the sum of client-facility distances. In addition, we have capacity constraints associated with each facility in the input to indicate the maximum number of clients that can be assigned to that facility. We give a constant-factor approximation algorithm for this problem by rounding a linear programming relaxation. In line with the previous approximation algorithms for the same problem, this is a pseudo-approximation algorithm that violates the capacities by at most a factor of $(1 + \epsilon)$.

In the second problem, we have a set of jobs to be scheduled on a set of machines to minimize either the makespan of the schedule or the more general total weighted completion time. Jobs have precedence constraints between them. Each job also have a resource requirement and the total resource use of jobs running concurrently at any point in a feasible schedule should not exceed the global resource cap given in the input. We initially use a divide-and-conquer approach to obtain a $O(\log n)$-approximation for the minimum makespan objective on identical machines. Then, we generalize the result to weighted completion time objective and uniformly related machines by combining the initial divide-and-conquer method with linear programming relaxations for these variations. Furthermore, we adapt our algorithm as a solution to an emerging problem in High Performance Computing (HPC): scheduling precedence constrained jobs under a power cap. We implement our algorithm and compare it to state-of-the-art greedy methods in a simulation environment on a variety of precedence relationships and power/performance data collected from real HPC applications. We find that our divide-and-conquer method improves performance by up to 75% compared to greedy scheduling algorithms.

CHAPTER 1

INTRODUCTION

In many combinatorial optimization problems, finding an optimum solution is NP-Hard. We turn to approximation algorithms for solutions with provable guarantees to these problems. An algorithm is said to have an approximation ratio of $\alpha > 1$ for such a minimization (resp. maximization) problem if it returns a solution of value $\alpha \times OPT$ (resp. $\frac{OPT}{\alpha}$) when the value of an optimal solution is OPT. We study two such optimization problems in this thesis: 1) Capacitated k-Median, and 2) Scheduling with Resource and Precedence Constraints.

In the classical k-Median problem, the input consists of a set of facilities and a set of clients on a metric space and a number k. We are asked to open a subset of at most k facilities and assign each client to an open facility. The goal is to minimize the total distance of connections between clients and their assigned facilities. We study the *Capacitated* k-Median problem where we have the additional constraints limiting the number of clients that can be connected to each facility by the capacity of that facility specified in the input. Existing constant-factor approximation algorithms for the Capacitated k-Median problem are all pseudo-approximations that violate either the capacities or the upper bound k on the number of open facilities. Using the natural linear program relaxation for the problem, one can only hope to get the violation factor down to 2. Li [61] introduced a novel LP to go beyond the limit of 2 and gave a constant-factor approximation algorithm that opens $(1+\epsilon)k$ facilities. We use the configuration LP of Li [61] to give a constant-factor approximation for the Capacitated k-Median problem in a seemingly harder configuration: we violate only the capacities by $1 + \epsilon$ [36]. This result settles the problem as far as pseudo-approximation algorithms are concerned.

The second problem we study is Scheduling under simultaneous Resource and Precedence Constraints. We are given a set $\mathcal{J}$ of n jobs to be scheduled on m parallel machines. Each job $j \in \mathcal{J}$ has a processing time $p_j \in \mathbb{Z}_{\geq 0}$ and a resource requirement $s_j \in \mathbb{Z}_{\geq 0}$. A global resource capacity $S \in \mathbb{Z}_{\geq 0}$ is given as a part of the input. The sum of the resource

requirements of the jobs running on the m machines at any point in time must be at most S. Moreover, we have precedence constraints given by a partial order $\prec$ on the jobs such that if $j \prec j'$ then j must complete before j' can start. A feasible schedule needs to run each job j on one of the machines for p_j amount of time to completion without interruption; i.e. the schedule needs to be non-preemptive. Precedence constrained scheduling and resource constrained scheduling are two central problems in scheduling theory. They are both studied extensively and their approximability status is well understood. However, we do not have an algorithm with nontrivial bounds for the combination problem where both constraints are imposed at the same time. We propose a two-step algorithm for the basic variation of this combination problem on identical machines with minimum makespan objective. We handle the precedence constraints in the first step using the well-known list scheduling algorithm. We stretch out this intermediate schedule in the second step to also satisfy the resource constraints by using a divide-and-conquer approach. Then, we use this algorithm as a subroutine and combine it with new linear programming relaxations to obtain similar results for the variations with the minimum total weighted completion time objective and uniformly related machines [35]. Finally, we demonstrate by empirical results that our algorithm is better than state-of-the-art greedy schedulers for our main motivating problem in the High Performance Computing space: precedence constrained scheduling under a power cap [37].

Chapter 2 is dedicated to Capacitated k-Median problem and and we present our results about Scheduling with Resource and Precedence Constraints in Chapter 3.

CHAPTER 2

CAPACITATED K-MEDIAN

2.1 Introduction

In the capacitated k-median problem (CKM), we are given a set F of facilities together with their capacities $u_i \in \mathbb{Z}_{>0}$ for $i \in F$, a set C of clients, a metric d on $F \cup C$, and a number k. We are asked to open some of these facilities $F' \subseteq F$ and give an assignment $\sigma : C \to F'$ connecting each client to one of the open facilities so that the number of open facilities is not bigger than k, i.e. $\left|F'\right| \leq k$ (*cardinality constraint*), and each facility $i \in F'$ is connected to at most u_i clients, i.e. $\left|\sigma^{-1}(i)\right| \leq u_i$ (*capacity constraint*). The goal is to minimize the sum of the connection costs, i.e. $\sum_{j \in C} d(\sigma(j), j)$.

Without the capacity constraint, i.e. $u_i = \infty$ for all $i \in F$, this is the famous k-median problem (KM) and it is NP-hard even on Euclidean plane [67]. Furthermore, it is NP-hard to approximate the problem within a factor of $1 + 2/\mathrm{e}$ [46]. On tree metrics, however, KM is efficiently solvable [52]. As another example of a special case, there is a polynomial-time approximation scheme[1] for KM on Euclidean space [6, 54]. As for the KM on general metrics, Lin and Vitter [65] gave an algorithm that outputs $(1 + \epsilon)k$ open facilities and stays within a constant factor of the optimal solution. The first approximation algorithm that outputs a feasible solution is the combination of Bartal's result about embedding arbitrary metrics into trees [15] with the fact that KM is solvable on trees. This yielded an approximation factor of $O(\log n \log \log n)$ which was later improved to $O(\log k \log \log k)$ by [28]. The first constant-factor approximation algorithm for KM is given by Charikar et al. [27] building upon the techniques of [64], guaranteeing a solution within $6\frac{2}{3}$ times the cost of the optimal solution. Then the approximation ratio has been improved by a series of papers [47, 26, 7, 46, 63, 21]. Jain and Vazirani [47] used the primal-dual schema and Lagrangian relaxation, and KM's

1. i.e. an algorithm that produces an output within a factor $1 + \epsilon$ of the value of the optimal solution for any given ϵ.

close connection to the Facility Location problem where we have costs for opening facilities instead of the cardinality constraint $\left|F'\right| \leq k$ and the objective is to minimize the sum of total connection cost and total opening cost. [7] analyzes several local search algorithms. [63] uses a pseudo-approximation algorithm that may violate the cardinality constraint to come up with a true approximation algorithm for KM with better approximation factor. The current best approximation factor for KM is $2.675 + \epsilon$ due to Byrka et al. [21], which was obtained by improving a part of the algorithm given by Li and Svensson [63].

On the other hand, we don't have a true constant approximation for CKM. All known constant-factor results are pseudo-approximations which violate either the cardinality or the capacity constraint. Korupolu et al. [55] used local search technique to give a $O(1+\epsilon)$-factor approximation algorithm that opens $(12+17/\epsilon)k$ facilities and $O(1/\epsilon^3)$-factor approximation algorithm that opens $(5+\epsilon)k$ facilities. Aardal et al. [2] gave an algorithm which finds a $(7+\epsilon)$-approximate solution to CKM by opening at most $2k$ facilities, i.e. violating the cardinality constraint by a factor of 2. Guha [42] gave an algorithm with approximation ratio 16 for the more relaxed *uniform* CKM, where all capacities are the same, by connecting at most $4u$ clients to each facility, thus violating the capacity constraint by 4. Li [59] gave a constant-factor algorithm for uniform CKM with capacity violation of only $2+\epsilon$ by improving the algorithm in [27]. For non-uniform capacities, Chuzhoy and Rabani [31] gave a 40-approximation for CKM by violating the capacities by a factor of 50 using a mixture of primal-dual schema and lagrangian relaxations. Their algorithm is for a slightly relaxed version of the problem called *soft* CKM where one is allowed to open multiple collocated copies of a facility in F. The CKM definition we gave above is sometimes referred to as *hard* CKM as opposed to this version. Recently, Byrka et al. [20] gave a constant-factor algorithm for hard CKM by keeping capacity violation factor under $3+\epsilon$.

All these algorithms for CKM use the basic LP relaxation for the problem which is known to have an unbounded integrality gap even when we are allowed to violate either the capacity or the cardinality constraint by $2-\epsilon$. In this sense, results of [2] and [59] can be considered

as reaching the limits of the basic LP relaxation in terms of restricting the violation factor. In order to go beyond these limits, Li [60] introduced a novel LP called the *rectangle* LP and presented a constant-factor approximation algorithm for soft uniform CKM by opening $(1+\epsilon)k$ facilities. This was later generalized by the same author to non-uniform CKM [61], where he introduced an even stronger LP relaxation called the *configuration* LP. Very recently, independently of the work in this thesis, Byrka et al. [23] used this configuration LP to give a similar algorithm for uniform CKM violating the capacities by $1+\epsilon$.

2.1.1 Our Result

In this work, we use the configuration LP of [61] to give an $O(1/\epsilon^5)$-approximation algorithm for non-uniform hard CKM which respects the cardinality constraint and connects at most $(1+\epsilon)u_i$ clients to any open facility $i \in F$. The running time of our algorithm is $n^{O(1/\epsilon)}$. Thus, with this result, we now have settled the CKM problem from the view of pseudo-approximation algorithms: either $(1+\epsilon)$-cardinality violation or $(1+\epsilon)$-capacity violation is sufficient for a constant approximation for CKM.

The known results for the CKM problem have suggested that designing algorithms with capacity violation (satisfying the cardinality constraint) is harder than designing algorithms with cardinality violation. Note, for example, that the best known cardinality violation factor for non-uniform CKM among algorithms using only the basic LP relaxation (a factor of 2 in [2]) matches the smallest possible cardinality violation factor dictated by the gap instance. In contrast, the best capacity-violation factor is $3+\epsilon$ due to [20], but the gap instance for the basic LP with the largest known gap eliminates only the algorithms with capacity violation smaller than 2.

Furthermore, we can argue that, for algorithms based on the basic LP and the configuration LP, a β-capacity violation can be converted to a β-cardinality violation, suggesting that allowing capacity violation is more restrictive than allowing cardinality violation. Suppose we have an α-approximation algorithm for CKM that violates the capacity constraint by a

factor of β, based on the basic LP relaxation. Given a solution (x, y) to the basic LP for a given CKM instance I, we construct a new instance I' by scaling k by a factor of β and scaling all capacities by a factor of $1/\beta$ (in a valid solution, we allow connections to be fractional, thus fractional capacities do not cause issues). Then it is easy to see that $(x, \beta y)$ is a valid LP solution to I' (with soft capacities). A solution to I' that only violates the capacity constraint by a factor of β is a solution to I that only violates the cardinality constraint by a factor of β. Thus, by considering the new instance, we conclude that for algorithms based on the basic LP relaxation, violating the cardinality constraint gives more power. The same argument can be made for algorithms based on the configuration LP: one can show that a valid solution to the configuration LP for I yields a valid solution to the configuration LP for I'. However, this reduction in the other direction does not work: due to constraint (2.3), scaling y variables by a factor of $1/\beta$ does not yield a valid LP solution.

Our Techniques. Our algorithm uses the configuration LP introduced in [61] and the framework of [61] that creates a two-level clustering of facilities. [61] considered the $(1+\epsilon)$-cardinality violation setting, which is more flexible in the sense that one has the much freedom to distribute the ϵk extra facilities. In our $(1+\epsilon)$-capacity violation setting, each facility i can provide an extra ϵu_i capacity; however, these extra capacities are restricted by the locations of the facilities. In particular, we need one more level of clustering to form so-called "groups" so that each group contains $\Omega(1/\epsilon)$ fractional open facility. Only with groups of $\Omega(1/\epsilon)$ facilities, we can benefit from the extra capacities given by the $(1+\epsilon)$-capacity scaling. Our algorithm then constructs distributions of local solutions. Using a dependent rounding procedure we can select a local solution from each distribution such that the solution formed by the concatenation of local solutions has a small cost. This initial solution may contain more than k facilities. We then remove some already-open facilities, and bound the cost incurred due to the removal of open facilities. When we remove a facility, we are guaranteed that there is a close group containing $\Omega(1/\epsilon)$ open facilities and the extra capacities provided by these facilities can compensate for the capacity of the removed facility.

Organization. In Sections 2.2 and 2.3, we describe the configuration LP introduced in [61] and our three-level clustering procedure respectively. In Section 2.4, we show how to construct the distributions of local solutions. Then finally in Section 2.5, we show how to obtain our final solution by combining the distributions we constructed.

2.2 The Basic LP and the Configuration LP

In this section, we give the configuration LP of [61] for CKM. We start with the following basic LP relaxation:

$$\min \quad \textstyle\sum_{i\in F, j\in C} d(i,j)x_{i,j} \qquad \text{s.t.} \qquad \text{(Basic LP)}$$

$$\textstyle\sum_{i\in F} y_i \le k; \qquad (2.1)$$

$$\textstyle\sum_{i\in F} x_{i,j} = 1, \quad \forall j \in C; \qquad (2.2)$$

$$x_{i,j} \le y_i, \quad \forall i \in F, j \in C; \qquad (2.3)$$

$$\textstyle\sum_{j\in C} x_{i,j} \le u_i y_i, \quad \forall i \in F; \qquad (2.4)$$

$$0 \le x_{i,j}, y_i \le 1, \quad \forall i \in F, j \in C. \qquad (2.5)$$

In the LP, y_i indicates whether a facility $i \in F$ is open, and $x_{i,j}$ indicates whether client $j \in C$ is connected to facility $i \in F$. Constraint (2.1) is the cardinality constraint assuring that the number of open facilities is no more than k. Constraint (2.2) says that every client must be fully connected to facilities. Constraint (2.3) requires a facility to be open in order to connect clients. Constraint (2.4) is the capacity constraint.

It is well known that the basic LP has unbounded integrality gap, even if we are allowed to violate the cardinality constraint or the capacity constraint by a factor of $2 - \epsilon$. In the gap instance for the capacity-violation setting, each facility has capacity u, k is $2u - 1$, and the metric consists of u isolated groups each of which has 2 facilities and $2u - 1$ clients that are all collocated. In other words, the distances within a group are all 0 but the distances between groups are nonzero. Any integral solution for this instance has to have a group with at most one open facility. Therefore, even with $(2 - 2/u)$-capacity-violation, we have to connect 1 client in this group to open facilities in other groups. On the other hand, a

fractional solution to the basic LP relaxation opens $2-1/u$ facilities in each group and serves the demand of each group using only the facilities in that group. Note that the gap instance disappears if we allow a capacity violation of 2.[2]

In order to overcome the gap in the cardinality-violation setting, Li [61] introduced a novel LP for CKM called the configuration LP, which we formally state below. Let us fix a set $B \subseteq F$ of facilities. Let $\ell = \Theta(1/\epsilon)$ and $\ell_1 = \Theta(\ell)$ be sufficiently large integers. Let $\mathcal{S} = \{S \subseteq B : |S| \leq \ell_1\}$ and $\widetilde{\mathcal{S}} = \mathcal{S} \cup \{\perp\}$, where $\perp$ stands for "any subset of B with size more than ℓ_1"; for convenience, we also treat $\perp$ as a set such that $i \in \perp$ holds for every $i \in B$. For $S \in \mathcal{S}$, let z^B_S indicate the event that the set of open facilities in B is exactly S and $z^B_\perp$ indicate the event that the number of open facilities in B is more than ℓ_1.

For every $S \in \widetilde{\mathcal{S}}$ and $i \in S$, $z^B_{S,i}$ indicates the event that $z^B_S = 1$ and i is open. (If $i \in B$ but $i \notin S$, then the event will not happen.) Notice that when $i \in S \neq \perp$, we always have $z^B_{S,i} = z^B_S$; we keep both variables for notational purposes. For every $S \in \widetilde{\mathcal{S}}, i \in S$ and client $j \in C$, $z^B_{S,i,j}$ indicates the event that $z^B_{S,i} = 1$ and j is connected to i. In an integral solution, all the above variables are $\{0,1\}$ variables. The following constraints are valid. To help understand the constraints, it is good to think of $z^B_{S,i}$ as $z^B_S \cdot y_i$ and $z^B_{S,i,j}$ as $z^B_S \cdot x_{i,j}$.

2. A similar instance can be given to show that the gap is still unbounded when the cardinality constraint is violated, instead of the capacity constraint, by less than 2: let $k = u + 1$ and each group have 2 facilities and $u + 1$ clients.

$$\sum_{S \in \widetilde{\mathcal{S}}} z^B_S = 1; \tag{2.6}$$

$$\sum_{S \in \widetilde{\mathcal{S}}: i \in S} z^B_{S,i} = y_i, \qquad \forall i \in B; \tag{2.7}$$

$$\sum_{S \in \widetilde{\mathcal{S}}: i \in S} z^B_{S,i,j} = x_{i,j}, \qquad \forall i \in B, j \in C; \tag{2.8}$$

$$z^B_{S,i} = z^B_S, \qquad \forall S \in \mathcal{S}, i \in S; \tag{2.9}$$

$$\sum_{i \in S} z^B_{S,i,j} \leq z^B_S, \qquad \forall S \in \widetilde{\mathcal{S}}, j \in C; \tag{2.10}$$

$$\sum_{j \in C} z^B_{S,i,j} \leq u_i z^B_{S,i}, \qquad \forall S \in \widetilde{\mathcal{S}}, i \in S; \tag{2.11}$$

$$\sum_{i \in B} z^B_{\perp,i} \geq \ell_1 z^B_{\perp}. \tag{2.12}$$

$$0 \leq z^B_{S,i,j} \leq z^B_{S,i} \leq z^B_S, \qquad \forall S \in \widetilde{\mathcal{S}}, i \in S, j \in C; \tag{2.13}$$

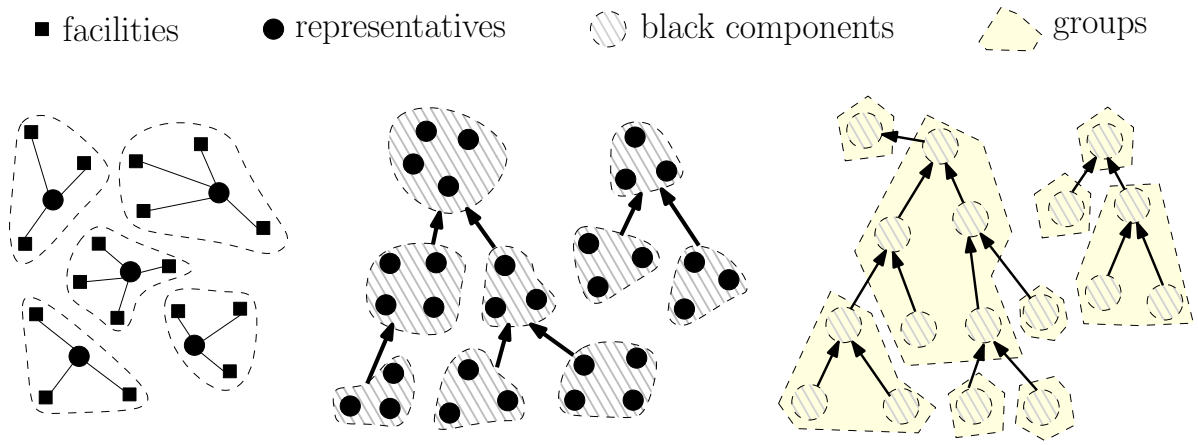
Constraint (2.6) says that $z^B_S = 1$ for exactly one $S \in \widetilde{\mathcal{S}}$. Constraint (2.7) says that if i is open then there is exactly one $S \in \widetilde{\mathcal{S}}$ with $z^B_{S,i} = 1$. Constraint (2.8) says that if j is connected to i then there is exactly one $S \in \widetilde{\mathcal{S}}$ such that $z^B_{S,i,j} = 1$. Constraint (2.13) is by the definition of variables. Constraint (2.9) holds as we mentioned earlier. Constraint (2.10) says that if $z^B_S = 1$ then j can be connected to at most 1 facility in S. Constraint (2.11) is the capacity constraint. Constraint (2.12) says that if $z^B_{\perp} = 1$, there are at least ℓ_1 open facilities in B.

The configuration LP is obtained from the basic LP by adding the z variables and Constraints (2.6) to (2.13) for every $B \subseteq F$. Since there are exponentially many subsets $B \subseteq F$, we don't know how to solve this LP efficiently. However, note that there are only polynomially many ($n^{O(\ell_1)}$) z^B variables for a fixed $B \subseteq F$. Given a fractional solution (x, y) to the basic LP relaxation, we can construct the values of z^B variables and check their feasibility for Constraints (2.6) to (2.13) in polynomial time as in [61]. Our rounding algorithm either constructs an integral solution with the desired properties, or outputs a set $B \subseteq F$ such that

Constraints (2.6) to (2.13) are infeasible. In the latter case, we can find a constraint in the configuration LP that (x, y) does not satisfy. Then we can run the ellipsoid method and the rounding algorithm in an iterative way (see, e.g., [24, 5]).

Notation From now on, we fix a solution $\left(\left\{x_{i,j} : i \in F, j \in C\right\}, \{y_i : i \in F\}\right)$ to the basic LP. We define $d_{\mathsf{av}}(j) := \sum_{i \in F} x_{i,j} d(i,j)$ to be the connection cost of j, for every $j \in C$. Let $D_i := \sum_{j \in C} x_{i,j} \left(d(i,j) + d_{\mathsf{av}}(j)\right)$ for every $i \in F$, and $D_S := \sum_{i \in S} D_i$ for every $S \subseteq F$. We denote the value of the solution (x, y) by $\mathsf{LP} := \sum_{i \in F, j \in C} x_{i,j} d(i,j) = \sum_{j \in C} d_{\mathsf{av}}(j)$. Note that $D_F = \sum_{i \in F, j \in C} x_{i,j} \left(d(i,j) + d_{\mathsf{av}}(j)\right) = \sum_{i \in F, j \in C} x_{i,j} d(i,j) + \sum_{j \in C} d_{\mathsf{av}}(j) \sum_{i \in F} x_{i,j} = 2\mathsf{LP}$. For any set $F' \subseteq F$ of facilities and $C' \subseteq C$ of clients, we shall let $x_{F',C'} := \sum_{i \in F', j \in C'} x_{i,j}$; we simply use $x_{i,C'}$ for $x_{\{i\},C'}$ and $x_{F',j}$ for $x_{F',\{j\}}$. For any $F' \subseteq F$, let $y_{F'} := \sum_{i \in F'} y_i$. Let $d(A, B) := \min_{i \in A, j \in B} d(i,j)$ denote the minimum distance between A and B, for any $A, B \subseteq F \cup C$; we simply use $d(i, B)$ for $d(\{i\}, B)$.

Moving of Demands After the set of open facilities is decided, the optimum connection assignment from clients to facilities can be computed by solving the minimum cost b-matching problem. Due to the integrality of the matching polytope, we may allow the connections to be fractional. That is, if there is a good fractional assignment, then there is a good integral assignment. So we can use the following framework to design and analyze the rounding algorithm. Initially there is one unit of demand at each client $j \in C$. During the course of our algorithm, we move demands fractionally within $F \cup C$; moving α units of demand from i to j incurs a cost of $\alpha d(i,j)$. At the end, all the demands are moved to F and each facility $i \in F$ has at most $(1 + O(\frac{1}{\ell}))u_i$ units of demand. We open a facility if it has positive amount of demand. Our goal is to bound the total moving cost by $O(\ell^5)\mathsf{LP}$ and the number of open facilities by k.



(a). bundles $\{U_v\}_{v\in R}$ (b). black components $\mathcal{J}$ and forest $\Upsilon^*_{\mathcal{J}}$ (c). groups $\mathcal{G}$ and forest $\Upsilon_{\mathcal{G}}$

Figure 2.1: The three-phase clustering procedure. In the first phase (Figure (a)), we partition F into bundles, centered at the set R of representatives. In the second phase (Figure (b)), we partition R into a family $\mathcal{J}$ of black components and construct a degree-2 rooted forest over $\mathcal{J}$. In the third phase (Figure(c)), we partition $\mathcal{J}$ into a family $\mathcal{G}$ of groups; $\Upsilon_{\mathcal{G}}$ is formed from $\Upsilon^*_{\mathcal{J}}$ by contracting each group into a single node.

2.3 Representatives, Black Components, and Groups

Our algorithm starts with bundling facilities together with a three-phase process each of which creates bigger and bigger clusters. At the end, we have a nicely formed network of sufficiently big clusters of facilities. See Figure 2.1 for illustration of the three-phase clustering.

2.3.1 Representatives, Bundles, and Initial Moving of Demands

In the first phase, we use a standard approach to facility location problems ([64, 85, 27, 61]) to partition the facilities into *bundles* $\{U_v\}_{v\in R}$, where each bundle U_v is associated with a center $v \in C$ that is called a *representative* and $R \subseteq C$ is the set of representatives. Each bundle U_v has a total opening at least $1/2$.

Let $R = \emptyset$ initially. Repeat the following process until C becomes empty: we select the client $v \in C$ with the smallest $d_{\mathsf{av}}(v)$ and add it to R; then we remove all clients j such that $d(j, v) \le 4d_{\mathsf{av}}(j)$ from C (thus, v itself is removed). We use v and its variants to index

representatives, and j and its variants to index general clients. The family $\{U_v : v \in R\}$ is the Voronoi diagram of F with R being the centers: let $U_v = \emptyset$ for every $v \in R$ initially; for each location $i \in F$, we add i to U_v for $v \in R$ that is closest to i. For any subset $V \subseteq R$, we use $U(V) := \bigcup_{v \in V} U_v$ to denote the union of Voronoi regions with centers V.

Lemma 2.3.1. *The following statements hold:*

(2.3.1a) *for all $v, v' \in R, v \neq v'$, we have $d(v, v') > 4 \max\{d_{\mathsf{av}}(v), d_{\mathsf{av}}(v')\}$*

(2.3.1b) *for all $j \in C$, there exists $v \in R$, such that $d_{\mathsf{av}}(v) \leq d_{\mathsf{av}}(j)$ and $d(v, j) \leq 4d_{\mathsf{av}}(j)$;*

(2.3.1c) *$y_{U_v} \geq 1/2$ for every $v \in R$;*

(2.3.1d) *for any $v \in R$, $i \in U_v$, and $j \in C$, we have $d(i, v) \leq d(i, j) + 4d_{\mathsf{av}}(j)$.*

Proof. First consider Property (2.3.1a). Assume $d_{\mathsf{av}}(v) \leq d_{\mathsf{av}}(v')$. When we add v to R, we remove all clients j satisfying $d(v, j) \leq 4d_{\mathsf{av}}(j)$ from C. If $v' \in R$, then it must have been $d(v, v') > 4d_{\mathsf{av}}(v')$. For Property (2.3.1b), just consider the iteration in which j is removed from C. The representative v added to R in this iteration satisfy the property. Then consider Property (2.3.1c). By Property (2.3.1a), we have $B := \{i \in F : d(i, v) \leq 2d_{\mathsf{av}}(v)\} \subseteq U_v$. Since $d_{\mathsf{av}}(v) = \sum_{i \in F} x_{i,v} d(i, v)$ and $\sum_{i \in F} x_{i,v} = 1$, we have $d_{\mathsf{av}}(v) \geq (1 - x_{B,v}) 2d_{\mathsf{av}}(v)$, implying $y_{U_v} \geq y_B \geq x_{B,v} \geq 1/2$, due to Constraint (2.3).

Then we consider Property (2.3.1d). By Property (2.3.1b), there is a client $v' \in R$ such that $d_{\mathsf{av}}(v') \leq d_{\mathsf{av}}(j)$ and $d(v', j) \leq 4d_{\mathsf{av}}(j)$. Since $d(i, v) \leq d(i, v')$ as $v' \in R$ and i was added to U_v, we have $d(i, v) \leq d(i, v') \leq d(i, j) + d(j, v') \leq d(i, j) + 4d_{\mathsf{av}}(j)$. □

The next lemma shows that moving demands from facilities to their corresponding representative doesn't cost much.

Lemma 2.3.2. *For every $v \in R$, we have $\sum_{i \in U_v} x_{i,C} d(i, v) \leq O(1) D_{U_v}$.*

Proof. By Property (2.3.1d), we have $d(i, v) \leq d(i, j) + 4d_{\mathsf{av}}(j)$ for every $i \in U_v$ and $j \in C$. Thus,

$$\sum_{i \in U_v} x_{i,C} d(i, v) \leq \sum_{i \in U_v, j \in C} x_{i,j} \big(d(i, j) + 4d_{\mathsf{av}}(j)\big) \leq \sum_{i \in U_v} 4D_i = 4D_{U_v}.$$

□

Since $\{U_v : v \in R\}$ forms a partition of F, we get the following corollary.

Corollary 2.3.3. $\sum_{v \in R, i \in U_v} x_{i,C} d(i,v) \leq O(1)\mathsf{LP}$.

Initial Moving of Demands With this corollary, we now move all the demands from C to R. First for every $j \in C$ and $i \in F$, we move $x_{i,j}$ units of demand from j to i. The moving cost of this step is exactly LP. After the step, all demands are at F and every $i \in F$ has $x_{i,C}$ units of demand. Then, for every $v \in R$ and $i \in U_v$, we move the $x_{i,C}$ units of demand at i to v. The moving cost for this step is $O(1)\mathsf{LP}$. Thus, after the initial moving, all demands are at the set R of representatives: a representative v has $x_{U_v,C}$ units of demand.

2.3.2 Black Components

In the second phase, we employ the minimum-spanning-tree construction of [61] to partition the set R of representatives into a family $\mathcal{J}$ of so-called *black components*. There is a degree-2 rooted forest $\Upsilon^*_{\mathcal{J}}$ over $\mathcal{J}$ with many good properties. For example, each non-root black component is not far away from its parent, and each root black component of $\Upsilon^*_{\mathcal{J}}$ contains a total opening of $\Omega(\ell)$. (For simplicity, we say the total opening at a representative $v \in R$ is y_{U_v}, which is the total opening at the bundle U_v.) The forest in [61] can have a large degree, while our algorithm requires the forest to have degree 2. This property is guaranteed by using the left-child-right-sibling representation.

We now describe the framework of [61]. We run the classic Kruskal's algorithm to find the minimum spanning tree MST of the metric (R, d), and then color the edges in MST in black, grey or white. In Kruskal's algorithm, we maintain the set E_{MST} of edges added to MST so far and a partition $\mathfrak{P}$ of R. Initially, we have $E_{\mathsf{MST}} = \emptyset$ and $\mathfrak{P} = \{\{v\} : v \in R\}$. The length of an edge $e \in \binom{R}{2}$ is the distance between the two endpoints of e. We sort all edges in $\binom{R}{2}$ in the ascending order of their lengths, breaking ties arbitrarily. For each pair

(v, v') in this order, if v and v' are not in the same partition in $\mathfrak{P}$, we add the edge (v, v') to E_{MST} and merge the two partitions containing v and v' respectively.

We then color edges in E_{MST}. For every $v \in R$, we say the *weight* of v is y_{U_v}; so every representative $v \in R$ has weight at least $1/2$ by Property (2.3.1c). For a subset $J \subseteq R$ of representatives, we say J is big if the weight of J is at least ℓ, i.e, $y_{U(J)} \geq \ell$; we say J is small otherwise. For any edge $e = (v, v') \in E_{\mathsf{MST}}$, we consider the iteration in Kruskal's algorithm in which the edge e is added to MST. After the iteration we merged the partition J_v containing v and the partition $J_{v'}$ containing v' into a new partition $J_v \cup J_{v'}$. If both J_v and $J_{v'}$ are small, then we call e a black edge. If J_v is small and $J_{v'}$ is big, we call e a grey edge, directed from v to v'; similarly, if $J_{v'}$ is small and J_v is big, e is a grey edge directed from v' to v. If both J_v and $J_{v'}$ are big, we say e is a white edge. So, we treat black and white edges as undirected edges and grey edges as directed edges.

We define a black component of MST to be a maximal set of vertices connected by black edges. Let $\mathcal{J}$ be the set of all black components. Thus $\mathcal{J}$ indeed forms a partition of R. We contract all the black edges in MST and remove all the white edges. The resulting graph is a forest $\Upsilon_{\mathcal{J}}$ of trees over black components in $\mathcal{J}$. Each edge is a directed grey edge. Later in Lemma 2.3.4, we show that the grey edges are directed towards the roots of the trees. For every component $J \in \mathcal{J}$, we define $L(J) := d(J, R \setminus J)$ to be the shortest distance between any representative in J and any representative not in J.

A component J in the forest $\Upsilon_{\mathcal{J}}$ may have many child-components. To make the forest binary, we use the left-child-right-sibling binary-tree representation of trees. To be more specific, for every component J', we sort all its child-components J according to non-decreasing order of $L(J)$. We add a directed edge from the first child to J' and a directed edge between every two adjacent children in the ordering, from the child appearing later in the ordering to the child appearing earlier. Let $\Upsilon^*_{\mathcal{J}}$ be the new forest. $\Upsilon^*_{\mathcal{J}}$ naturally defines a new child-parent relationship between components.

Lemma 2.3.4. *$\mathcal{J}$ and $\Upsilon^*_{\mathcal{J}}$ satisfy the following properties:*

(2.3.4a) *for every $J \in \mathcal{J}$, there is a spanning tree over the representatives in J such that for every edge (v, v') in the spanning tree we have $d(v, v') \leq L(J)$;*

(2.3.4b) *every root component $J \in \mathcal{J}$ of $\Upsilon^*_{\mathcal{J}}$ has $y_{U(J)} \geq \ell$ and every non-root component $J \in \mathcal{J}$ has $y_{U(J)} < \ell$;*

(2.3.4c) *every root component $J \in \mathcal{J}$ of $\Upsilon^*_{\mathcal{J}}$ has either $y_{U(J)} < 2\ell$ or $|J| = 1$;*

(2.3.4d) *for any non-root component J and its parent J', we have $L(J) \geq L(J')$;*

(2.3.4e) *for any non-root component J and its parent J', we have $d(J, J') \leq O(\ell)L(J)$;*

(2.3.4f) *every component J has at most two children.*

The rest of the section is dedicated to the proof of Lemma 2.3.4. We first prove some of the above properties for *the original forest* $\Upsilon_{\mathcal{J}}$. We show that all black edges between the representatives in J are considered before all the edges in $J \times (R \setminus J)$ in Kruskal's algorithm. Assume otherwise. Consider the first edge e in $J \times (R \setminus J)$ we considered. Before this iteration, J is not connected yet. Then we add e to the minimum spanning tree; since J is a black component, e is gray or white. In either case, the new partition J' formed by adding e will have weight more than ℓ. This implies all edges in $J' \times (R \setminus J')$ added later to the MST are not black. Moreover, $J \setminus J', J' \setminus J$ and $J \cap J'$ are all non-empty. This contradicts the fact that J is a black component. Therefore, all black edges in J has length at most $L(J)$, implying Property (2.3.4a) .

Focus on a tree T in the initial forest $\Upsilon_{\mathcal{J}}$ and any small black component J in T. All black edges between the representatives in J are added to MST before any edge in $J \times (R \setminus J)$. The first edge in $J \times (R \setminus J)$ added to MST is a grey edge directed from J to some other black component: it is not white because J is small; it is not black since J is a black component. Thus, it is a grey edge in T. Therefore, the growth of the tree T in Kruskal's algorithm is as follows. The first grey edge in T is added between two black components, one of them is big and the other is small. We define the root of T to be the big component. At each time, we add a new small black component J to the current T via a grey edge directed from J to T.

(During this process, white edges incident to T may be added.) So, the tree T is a rooted tree with grey edges, where all edges are directed towards the root. So, Property (2.3.4b) holds. Moreover, the length of the grey edge between J and its parent J' is $d(J, J') = L(J)$, which is stronger than Property (2.3.4e). Since $d(J, J') \geq d(J', R \setminus J') = L(J')$, we have Property (2.3.4d).

The root J of T is a big black component. Suppose it contains two or more representatives; so it's not a singleton. Consider the last black edge (v, v') added between J_v and $J_{v'}$ to make $J = J_v \cup J_{v'}$. Since (v, v') is a black edge, both J_v and $J_{v'}$ are small, i.e. $y_{U(J_v)}, y_{U(J_{v'})} < \ell$. Therefore, we have $y_{U(J)} = y_{U(J_v)} + y_{U(J_{v'})} < 2\ell$, proving Property (2.3.4c).

Now, we move on to prove all the properties of the lemma for *the final forest* $\Upsilon^*_{\mathcal{J}}$. We used the left-child-right-sibling binary-tree representation of $\Upsilon_{\mathcal{J}}$ to obtain $\Upsilon^*_{\mathcal{J}}$. Thus , Property (2.3.4f) holds for $\Upsilon^*_{\mathcal{J}}$. Property (2.3.4a) is independent of the forest and thus still holds for $\Upsilon^*_{\mathcal{J}}$. A component is a root in $\Upsilon_{\mathcal{J}}$ if and only if it is a root in $\Upsilon^*_{\mathcal{J}}$. Thus, properties (2.3.4b) and (2.3.4c) are maintained for $\Upsilon^*_{\mathcal{J}}$. Since we sorted the children of a component according to L values before constructing the left-child-right-sibling binary tree, Property (2.3.4d) holds for $\Upsilon^*_{\mathcal{J}}$.

For every component J and its parent J' in the forest $\Upsilon^*_{\mathcal{J}}$, we have $L(J) = d(J, R \setminus J) = d(J, J'')$, where J'' is the parent of J in the initial forest $\Upsilon_{\mathcal{J}}$. J' is either J'', or a child of J'' in $\Upsilon_{\mathcal{J}}$. In the former case, we have $d(J, J') = d(J, J'') = L(J)$. In the latter case, we have that $d(J', J'') = L(J') \leq L(J) = d(J, J'')$. Due to Property (2.3.4a), we have a path connecting some representative in J to some representative in J', with internal vertices being representatives in J'', and all edges having length at most $L(J)$. Moreover, there are at most 4ℓ representatives in J'' due to Properties (2.3.4b), (2.3.4c), and (2.3.1c). Thus, we have $d(J, J') \leq O(\ell) d(J, J'') = O(\ell) L(J)$. Thus, Property (2.3.4e) holds for $\Upsilon^*_{\mathcal{J}}$. This finishes the proof of Lemma 2.3.4.

2.3.3 Groups

In the third phase, we apply a simple greedy algorithm to the forest $\Upsilon^*_{\mathcal{J}}$ to partition the set $\mathcal{J}$ of black components into a family $\mathcal{G}$ of *groups*, where each group $G \in \mathcal{G}$ contains many black components that are connected in $\Upsilon^*_{\mathcal{J}}$. By contracting each group $G \in \mathcal{G}$, the forest $\Upsilon^*_{\mathcal{J}}$ over the set $\mathcal{J}$ of black components becomes a forest $\Upsilon_{\mathcal{G}}$ over the set $\mathcal{G}$ of groups. Each group has a total opening of $\Omega(\ell)$, unless it is a leaf-group in $\Upsilon_{\mathcal{G}}$.

We partition the set $\mathcal{J}$ into groups using a technique similar to [20, 22]. For each rooted tree $T = (\mathcal{J}_T, E_T)$ in $\Upsilon^*_{\mathcal{J}}$, we construct a group G of black components as follows. Initially, let G contain the root component of T. While $\sum_{J \in G} y_{U(J)} < \ell$ and $G \neq \mathcal{J}_T$, repeat the following procedure. Choose the component $J \in \mathcal{J}_T \setminus G$ that is adjacent to G in T, with the smallest L-value, and add J to G.

Thus, by the construction G is connected in T. After we have constructed the group G, we add G to $\mathcal{G}$. We remove all black components in G from T. Then, each T is broken into many rooted trees; we apply the above procedure recursively for each rooted tree.

So, we have constructed a partition $\mathcal{G}$ for the set $\mathcal{J}$ of components. If for every $G \in \mathcal{G}$, we contract all components in G into a single node, then the rooted forest $\Upsilon^*_{\mathcal{J}}$ over $\mathcal{J}$ becomes a rooted forest $\Upsilon_{\mathcal{G}}$ over the set $\mathcal{G}$ of groups. $\Upsilon_{\mathcal{G}}$ naturally defines a parent-child relationship over $\mathcal{G}$. The following lemma uses Properties (2.3.4a) to (2.3.4f) of $\mathcal{J}$ and the way we construct $\mathcal{G}$.

Lemma 2.3.5. *The following statements hold for the set $\mathcal{G}$ of groups and the rooted forest $\Upsilon_{\mathcal{G}}$ over $\mathcal{G}$:*

(2.3.5a) *any root group $G \in \mathcal{G}$ contains a single root component $J \in \mathcal{J}$;*

(2.3.5b) *if $G \in \mathcal{G}$ is not a root group, then $\sum_{J \in G} y_{U(J)} < 2\ell$;*

(2.3.5c) *if $G \in \mathcal{G}$ is a non-leaf group, then $\sum_{J \in G} y_{U(J)} \geq \ell$;*

(2.3.5d) *let $G \in \mathcal{G}, G' \in \mathcal{G}$ be the parent of G, $J \in G$ and $v \in J$, then the distance between v and any representative in $\bigcup_{J' \in G'} J'$ is at most $O(\ell^2)L(J)$;*

(2.3.5e) *any group G has at most $O(\ell)$ children.*

Proof. For a root component J, we have $y_{U(J)} \geq \ell$ by Property (2.3.4b). Thus, any root group G contains a single root component J, which is exactly Property (2.3.5a).

When constructing the group G from the tree $T = (\mathcal{J}_T, E_T)$, the terminating condition is $G = \mathcal{J}_T$ or $\sum_{J \in G} y_{U(J)} \geq \ell$. Thus, if G is not a leaf-group, then the condition $G = \mathcal{J}_T$ does not hold; thus we have $\sum_{J \in G} y_{U(J)} \geq \ell$, implying Property (2.3.5c).

By Property (2.3.4b), any non-root component J has $y_{U(J)} < \ell$. Thus, if G is not a root group, the terminating condition constructing G implies that G had total weight less than ℓ right before the last black component was added to it. Then we have $\sum_{J \in G} y_{U(J)} < 2\ell$, implying Property (2.3.5b).

Now, consider Property (2.3.5d). From Property (2.3.4d), it is easy to see that the group G constructed from the tree $T = (\mathcal{J}_T, E_T)$ has the following property: the L value of any component in G is at most the L-value of any component in $\mathcal{J}_T \setminus G$. Let G be a non-root group and G' be its parent; let $J \in G$ and $J' \in G'$ be black components. Thus, there is a path in $\Upsilon^*_{\mathcal{J}}$ from J to J', where components have L-values at most $L(J)$. The edges in the path have length at most $O(\ell)L(J)$ by Property (2.3.4e). Moreover, Property (2.3.4a) implies that the representatives in each component in the path are connected by edges of length at most $L(J)$. Thus, we can find a path from v to v' that go through representatives in $\bigcup_{J'' \in G \cup G'} J''$, and every edge in the path has length at most $O(\ell)L(J) = O(\ell)d(J, R \setminus J)$. By Property (2.3.1c), (2.3.4b) and (2.3.4c), the total representatives in the components contained in G (as well as in G') is at most 4ℓ. Thus, the distance between v and v' is at most $O(\ell^2)L(J)$, which is exactly Property (2.3.5d).

Finally, since the forest $\Upsilon^*_{\mathcal{J}}$ is binary and every group $G \in \mathcal{G}$ contains at most $O(\ell)$ components, we have that every group G contains at most $O(\ell)$ children, implying Property (2.3.5e). □

2.4 Constructing Local Solutions

In this section, we shall construct a local solution, or a distribution of local solutions, for a given set $V \subseteq R$ which is the union of some black components. A local solution for V contains a pair $(S \subseteq U(V), \beta \in \mathbb{R}_{\geq 0}^{U(V)})$, where S is the facilities we open in $U(V)$ and β_i for each $i \in U(V)$ is the amount of supply at i: the demand that can be satisfied by i. Thus $\beta_i = 0$ if $i \in U(V) \setminus S$. We shall use the supplies at $U(V)$ to satisfy the $x_{U(V),C}$ demands at V after the initial moving of demands; thus, we require $\sum_{i \in U(V)} \beta_i = x_{U(V),C}$. There are two other main properties we need the distribution to satisfy: (a) the expected size of S from the distribution is not too big, and (b) the cost of matching the demands at V and the supplies at $U(V)$ is small.

We distinguish between *concentrated* black components and *non-concentrated* black components. Roughly speaking, a component $J \in \mathcal{J}$ is concentrated if in the fractional solution (x, y), for most clients $j \in C$, j is either almost fully served by facilities in $U(J)$, or almost fully served by facilities in $F \setminus U(J)$. We shall construct a distribution of local solutions for each concentrated component J. We require Constraints (2.6) to (2.12) to be satisfied for $B = U(J)$ (if not, we return the set $U(J)$ to the separation oracle) and let z^B be the vector satisfying the constraints. Roughly speaking, the z^B-vector defines a distribution of local solutions for V. A local solution (S, β) is good if S is not too big and the total demand $\sum_{i \in S} \beta_i$ satisfied by S is not too small. Then, our algorithm randomly selects (S, β) from the distribution defined by z^B, under the condition that (S, β) is good. The fact that J is concentrated guarantees that the total mass of good local solutions in the distribution is large; therefore the factors we lose due to the conditioning are small.

For non-concentrated components, we construct a single local solution (S, β), instead of a distribution of local solutions. Moreover, the construction is for the union V of some non-concentrated components, instead of an individual component. The components that comprise V are close to each other; by the fact that they are non-concentrated, we can move demands arbitrarily within V, without incurring too much cost. Thus we can essentially

treat the distances between representatives in V as 0. Then we are only concerned with two parameters for each facility $i \in U(V)$: the distance from i to V and the capacity u_i. Using a simple argument, the optimum fractional local solution (that minimizes the cost of matching the demands and supplies) is almost integral: it contains at most 2 fractionally open facilities. By fully opening the two fractional facilities, we find an integral local solution with small number of open facilities.

The remaining part of this section is organized as follows. We first formally define concentrated black components, and explain the importance of the definition. We then define the earth-mover-distance, which will be used to measure the cost of satisfying demands using supplies. The construction of local solutions for concentrated components and non-concentrated components will be stated in Theorem 2.4.4 and Lemma 2.4.9 respectively.

2.4.1 Concentrated Black Components and Earth Mover Distance

The definition of concentrated black component is the same as that of [61], except that we choose the parameter ℓ_2 differently.

Definition 2.4.1. *Define* $\pi_J = \sum_{j \in C} x_{U(J),j}(1 - x_{U(J),j})$, *for every black component* $J \in \mathcal{J}$. *A black component* $J \in \mathcal{J}$ *is said to be* concentrated *if* $\pi_J \leq x_{U(J),C}/\ell_2$, *and* non-concentrated *otherwise, where* $\ell_2 = \Theta(\ell^3)$ *is large enough.*

We use $\mathcal{J}^{\mathsf{C}}$ to denote the set of concentrated components and $\mathcal{J}^{\mathsf{N}}$ to denote the set of non-concentrated components. The next lemma from [61] shows the importance of π_J. For the completeness of the thesis, we include its proof here.

Lemma 2.4.2. *For any* $J \in \mathcal{J}$, *we have* $L(J)\pi_J \leq O(1)D_{U(J)}$.

Proof. Let $B = U(J)$. For every $i \in B, j \in C$, we have $d(i, J) \leq d(i, j) + 4d_{\mathsf{av}}(j)$ by

Property (2.3.1d) and the fact that $i \in U_v$ for some $v \in J$. Thus,

$$\begin{aligned}
&L(J)\pi(J) = L(J)\sum_{j\in C} x_{B,j}(1-x_{B,j}) = L(J)\sum_{j\in C, i\in B, i'\in F\setminus B} x_{i,j}x_{i',j} \\
&\le \sum_{i\in B, j\in C, i'\in F\setminus B} x_{i,j}x_{i',j}\cdot 2d(i',J) \le 2\sum_{i\in B,j\in C} x_{i,j}\sum_{i'\in F} x_{i',j}\big(d(i',j)+d(j,i)+d(i,J)\big) \\
&= 2\sum_{i\in B,j\in C} x_{i,j}\Big(d_{\mathsf{av}}(j)+d(j,i)+d(i,J)\Big) \le 2\sum_{i\in B,j\in C} x_{i,j}\Big(2d(i,j)+5d_{\mathsf{av}}(j)\Big) \\
&= 2\sum_{i\in B}(5D_i) = 10D_B.
\end{aligned}$$

The first inequality is by $L(J) \le 2d(i', J)$ for any $i' \in F \setminus B = U_{R\setminus J}$: $d(i', R \setminus J) \le d(i', J)$ implies $L(J) = d(R \setminus J, J) \le d(R \setminus J, i') + d(i', J) \le 2d(i', J)$. The second inequality is by triangle inequality and the third one is by $d(i, J) \le d(i, j) + 4d_{\mathsf{av}}(j)$. All the equalities are by simple manipulations of notations. □

Recall that $L(J) = d(J, R \setminus J)$ and $x_{U(J),C}$ is the total demand in J after the initial moving. Thus, according to Lemma 2.4.2, if J is not concentrated, we can use $D_{U(J)}$ to charge the cost for moving all the $x_{U(J),C}$ units of demand out of J, provided that the moving distance is not too big compared to $L(J)$. This gives us freedom for handling non-concentrated components. If J is concentrated, the amount of demand that is moved out of J must be comparable to π_J; this will be guaranteed by the configuration LP.

In order to measure the moving cost of satisfying demands using supplies, we define the earth mover distance:

Definition 2.4.3 (Earth Mover Distance)**.** *Given a set $V \subseteq R$ with $B = U(V)$, a demand vector $\alpha \in \mathbb{R}^V_{\ge 0}$ and a supply vector $\beta \in \mathbb{R}^B_{\ge 0}$ such that $\sum_{v\in V}\alpha_v \le \sum_{i\in B}\beta_i$, the earth mover distance from α to β is defined as $\mathrm{EMD}_V(\alpha,\beta) := \inf_f \sum_{v\in V, i\in B} f(v,i)d(v,i)$, where f is over all functions from $V \times B$ to $\mathbb{R}_{\ge 0}$ such that*

- *$\sum_{i\in B} f(v,i) = \alpha_v$ for every $v \in V$;*

- $\sum_{v \in V} f(v,i) \leq \beta_i$ *for every* $i \in B$.

For some technical reason, we allow some fraction of a supply to be unmatched. From now on, we shall use $\alpha_v = x_{U_v,C}$ to denote the amount of demand at v after the initial moving. For any set $V \subseteq R$ of representatives, we use $\alpha|_V$ to denote the vector α restricted to the coordinates in V.

2.4.2 Distributions of Local Solutions for Concentrated Components

In this section, we construct distributions for components in $\mathcal{J}^{\mathsf{C}}$, by proving:

Theorem 2.4.4. *Let* $J \in \mathcal{J}^{\mathsf{C}}$ *and let* $B = U(J)$. *Assume Constraints* (2.6) *to* (2.12) *are satisfied for* B. *Then, we can find a distribution* $(\phi_{S,\beta})_{S \subseteq B, \beta \in \mathbb{R}^B_{\geq 0}}$ *of pairs* (S,β), *such that*

(2.4.4a) $s_\phi := \mathbb{E}_{(S,\beta)\sim\phi} |S| \in [y_B, y_B(1 + 2\ell\pi_J/x_{B,C})]$, *and* $s_\phi = y_B$ *if* $y_B > 2\ell$,

and for every (S,β) *in the support of* ϕ, *we have*

(2.4.4b) $|S| \in \{\lfloor s_\phi \rfloor, \lceil s_\phi \rceil\}$;

(2.4.4c) $\beta_i \leq (1 + O(1/\ell))u_i$ *if* $i \in S$ *and* $\beta_i = 0$ *if* $i \in B \setminus S$;

(2.4.4d) $\sum_{i \in S} \beta_i = x_{B,C} = \sum_{v \in J} \alpha_v$.

Moreover, the distribution ϕ *satisfies*

(2.4.4e) *the support of* ϕ *has size at most* $n^{O(\ell)}$;

(2.4.4f) $\mathbb{E}_{(S,\beta)\sim\phi} \operatorname{EMD}_J(\alpha|_J, \beta) \leq O(\ell^4) D_B$.

To prove the theorem, we first construct a distribution ψ that satisfies most of the properties; then we modify it to obtain the final distribution ϕ. Notice that a typical black component J has $y_B \leq 2\ell$; however, when J is a root component containing a single representative, y_B might be very large. For now, let us just assume $y_B \leq 2\ell$. We deal with the case where $|J| = 1$ and $y_B > 2\ell$ at the end of this section.

Since Constraints (2.6) to (2.12) are satisfied for B, we can use the z^B variables satisfying these constraints to construct a distribution ζ over pairs $(\chi \in [0,1]^{B \times C}, \mu \in [0,1]^B)$, where μ

indicates the set of open facilities in B and χ indicates how the clients in J are connected to facilities in B. Let $\mathcal{S} = \{S \subseteq B : |S| \leq \ell_1\}$ and $\widetilde{\mathcal{S}} = S \cup \{\perp\}$ as in Section 2.2. For simplicity, for any $\mu \in [0,1]^B$, we shall use μ_B to denote $\sum_{i \in B} \mu_i$. For any $\chi \in [0,1]^{B \times C}, i \in B$ and $j \in C$, we shall use $\chi_{i,C}$ to denote $\sum_{j \in C} \chi_{i,j}$, $\chi_{B,j}$ to denote $\sum_{i \in B} \chi_{i,j}$, and $\chi_{B,C}$ to denote $\sum_{i \in B} \chi_{i,C} = \sum_{j \in C} \chi_{B,j}$.

The distribution ζ is defined as follows. Initially, let $\zeta_{\chi,\mu} = 0$ for all $\chi \in [0,1]^{B \times C}$ and $\mu \in [0,1]^B$. For each $S \in \widetilde{\mathcal{S}}$ such that $z_S^B > 0$, increase $\zeta_{\chi,\mu}$ by z_S^B for the χ, μ satisfying $\chi_{i,j} = z_{S,i,j}^B / z_S^B, \mu_i = z_{S,i}^B / z_S^B$ for every $i \in B, j \in C$. So, for every pair (χ, μ) in the support of ζ, we have $\chi_{i,j} \leq \mu_i, \chi_{i,C} \leq u_i \mu_i$ for every $i \in B, j \in C$. Moreover, either μ is integral, or $\mu_B \geq \ell_1$. Since $\sum_{S \in \widetilde{\mathcal{S}}} z_S^B = 1$, ζ is a distribution over pairs (χ, μ). It is not hard to see that $\mathbb{E}_{(\chi,\mu) \sim \zeta} \chi_{i,j} = x_{i,j}$ for every $i \in B, j \in C$ and $\mathbb{E}_{(\chi,\mu) \sim \zeta} \mu_i = y_i$ for every $i \in B$. The support of ζ has $n^{O(\ell)}$ size.

Definition 2.4.5. *We say a pair (χ, μ) is good if*

(2.4.5a) $\mu_B \leq y_B / (1 - 1/\ell)$;

(2.4.5b) $\chi_{B,C} \geq (1 - 1/\ell) x_{B,C}$.

We are only interested in good pairs in the support of ζ. We show that the total probability of good pairs in the distribution ζ is large. Let Ξ_{a} denote the set of pairs (χ, μ) satisfying Property (2.4.5a) and Ξ_{b} denote the set of pairs (χ, μ) satisfying Property (2.4.5b). Notice that $\mathbb{E}_{(\chi,\mu) \sim \zeta} \mu_B = y_B$. By Markov inequality, we have $\sum_{(\chi,\mu) \in \Xi_{\mathrm{a}}} \zeta_{\chi,\mu} \geq 1/\ell$. The proof of the following lemma uses elementary mathematical tools.

Lemma 2.4.6. $\sum_{(\chi,\mu) \notin \Xi_{\mathrm{b}}} \zeta_{\chi,\mu} \leq \ell \pi_J / x_{B,C}$.

Proof. The idea is to use the property that J is concentrated. To get some intuition, consider the case where $\pi_J = 0$. For every $j \in C$, either $x_{B,j} = 0$ or $x_{B,j} = 1$. Thus, all pairs (χ, μ) in the support of ζ have $\chi_{B,j} = x_{B,j}$ for every $j \in C$; thus $\chi_{B,C} = x_{B,C}$.

Assume towards contradiction that $\sum_{(\chi,\mu) \notin \Xi_{\mathrm{b}}} \zeta_{\chi,\mu} > \ell \pi_J / x_{B,C}$. We sort all pairs (χ, μ) in the support of ζ according to descending order of $\chi_{B,C}$. For any $t \in [0,1)$, and $j \in C$,

define $g_{t,j} \in [0,1]$ as follows. Take the first pair (χ,μ) in the ordering such that the total ζ value of the pairs (χ',μ') before (χ,μ) in the ordering plus $\zeta_{\chi,\mu}$ is greater than t. Then, define $g_{t,j} = \chi_{B,j}$ and define $g_t = \sum_{j\in C} g_{t,j} = \chi_{B,C}$.

Fix a client $j \in C$, we have

$$x_{B,j}(1-x_{B,j}) = \int_0^{x_{B,j}} (1-2t)\mathsf{d}t = \int_0^1 \mathbf{1}_{t<x_{B,j}}(1-2t)\mathsf{d}t \geq \int_0^1 g_{t,j}(1-2t)\mathsf{d}t,$$

where $\mathbf{1}_{t<x_{B,j}}$ is the indicator variable for the event that $t < x_{B,j}$. The inequality comes from the fact that $\int_0^1 \mathbf{1}_{t<x_{B,j}}\mathsf{d}t = x_{B,j} = \int_0^1 g_{t,j}\mathsf{d}t$, $g_{t,j} \in [0,1]$ for every $t \in [0,1)$, and $1-2t$ is a decreasing function of t.

Summing up the inequality over all $j \in C$, we have $\pi_J \geq \int_0^1 g_t(1-2t)\mathsf{d}t$. By our assumption that $\sum_{(\chi,\mu)\notin\Xi_\mathrm{b}} \zeta_{\chi,\mu} > \ell\pi_J/x_{B,C}$, there exists a number $t^* < 1-\ell\pi_J/x_{B,C}$ such that $g_t \leq (1-1/\ell)x_{B,C}$ for every $t \in [t^*,1)$. As g_t is a non-increasing function of g and $\int_0^1 g_t\mathsf{d}t = x_{B,C}$, it is not hard to see that $\int_0^1 g_t(1-2t)\mathsf{d}t$ is minimized when $g_t = (1-1/\ell)x_{B,C}$ for every $t \in [t^*,1)$ and $g_t = \frac{x_{B,C}-(1-1/\ell)x_{B,C}(1-t^*)}{t^*} = \frac{1/\ell+t^*-t^*/\ell}{t^*}x_{B,C}$ for every $t \in [0,t^*)$. We have

$$\begin{aligned}
\pi_J &\geq \left(\int_0^{t^*} \frac{1/\ell+t^*-t^*/\ell}{t^*}(1-2t)\mathsf{d}t + \int_{t^*}^1 (1-1/\ell)(1-2t)\mathsf{d}t\right) x_{B,C} \\
&= \left(\frac{1/\ell+t^*-t^*/\ell}{t^*}\left(t^*-(t^*)^2\right) - (1-1/\ell)\left(t^*-(t^*)^2\right)\right) x_{B,C} \\
&= \frac{1}{\ell t^*}\left(t^*-(t^*)^2\right) x_{B,C} = \frac{1-t^*}{\ell}x_{B,C} > \frac{\ell\pi_J/x_{B,C}}{\ell}x_{B,C} = \pi_J,
\end{aligned}$$

leading to a contradiction. Thus, we have that $\sum_{(\chi,\mu)\notin\Xi_\mathrm{b}} \zeta_{\chi,\mu} \leq \ell\pi_J/x_{B,C}$. This finishes the proof of Lemma 2.4.6. □

Overall, we have $Q := \sum_{(\chi,\mu)\text{ good}} \zeta_{\chi,\mu} = \sum_{(\chi,\mu)\in\Xi_\mathrm{a}\cap\Xi_\mathrm{b}} \zeta_{\chi,\mu} \geq 1/\ell - \ell\pi_J/x_{B,C} \geq 1/\ell - 1/(2\ell) = 1/(2\ell)$, where the second inequality used the fact that $\pi_J \leq x_{B,C}/(2\ell^2)$ for $J \in \mathcal{J}^\mathsf{C}$.

Now focus on each good pair (χ,μ) in the support of ζ. Since $J \in \mathcal{J}^\mathsf{C}$ and $(\chi,\mu) \in \Xi_\mathrm{a}$,

we have $\mu_B \le y_B/(1-1/\ell) \le 2\ell/(1-1/\ell) < \ell_1$ (since we assumed $y_B \le 2\ell$), if ℓ_1 is large enough. So, $\mu \in \{0,1\}^B$. Then, let $S = \{i \in B : \mu_i = 1\}$ be the set indicated by μ, and $\beta_i = \chi_{i,C}/(1-1/\ell)$ for every $i \in B$. For this (S,β), Property (2.4.4c) is satisfied, and we have $\sum_{i\in B}\beta_i = \chi_{B,C}/(1-1/\ell) \ge x_{B,C}$. We then set $\psi_{S,\beta} = \zeta_{\chi,\mu}/Q$. Thus, ψ indeed forms a distribution over pairs (S,β). Moreover, the support of ζ has size $n^{O(\ell)}$, so does the support of ψ. Thus Property (2.4.4e) holds.

Let $s_\psi := \mathbb{E}_{(S,\beta)\sim\psi}|S| = \mathbb{E}_{(\chi,\mu)\sim\zeta}\left[\mu_B \middle| (\chi,\mu) \text{ good}\right]$. Notice that $\mathbb{E}_{(\chi,\mu)\sim\zeta}\mu_B = y_B$. By Lemma 2.4.6, we have $\sum_{(\chi,\mu)\notin\Xi_{\mathrm{b}}}\zeta_{\chi,\mu} \le \ell\pi_J/x_{B,C}$. Thus, $\mathbb{E}_{(\chi,\mu)\sim\zeta}\left[\mu_B \middle| (\chi,\mu)\in\Xi_{\mathrm{b}}\right] \le y_B/(1-\ell\pi_j/x_{B,C})$. Since the condition $(\chi,\mu)\in\Xi_{\mathrm{a}}$ requires μ_B to be upper bounded by some threshold, $\mathbb{E}_{(\chi,\mu)\sim\zeta}\left[\mu_B \middle| (\chi,\mu)\in\Xi_{\mathrm{b}}\cap\Xi_{\mathrm{a}}\right]$ can only be smaller. Thus, we have that $s_\psi \le y_B/(1-\ell\pi_J/x_{B,C}) \le y_B(1+2\ell\pi_J/x_{B,C})$.

The proof of Property (2.4.4f) for ψ is long and tedious.For simplicity, we use $\hat{\mathbb{E}}[\cdot]$ to denote $\mathbb{E}_{(\chi,\mu)\sim\zeta}\left[\cdot \middle| (\chi,\mu) \text{ good}\right]$, and $a = 1/(1-1/\ell)$ to denote the scaling factor we used to define β. Indeed, we shall lose a factor $O(\ell^2)$ later and thus we shall prove Property (2.4.4f) for ψ with the $O(\ell^2)$ term on the right:

Lemma 2.4.7. $\hat{\mathbb{E}}[\mathrm{EMD}_J(\alpha|_J,\beta)] \le O(\ell^2)D_B$, *where β depends on χ as follows: $\beta_i = a\chi_{i,C}$ for every $i \in B$.*

Proof. Focus on a good pair (χ,μ) and the β it defined: $\beta_i = a\chi_{i,C}$ for every $i \in B$. We call α the demand vector and β the supply vector. Since (χ,μ) is good, $\sum_{i\in B}\beta_i = a\chi_{B,C} \ge x_{B,C} = \sum_{v\in J}\alpha_v$. Thus we can satisfy all the demands and $\mathrm{EMD}(\alpha,\beta)$ is not ∞.

We satisfy the demands in two steps. In the first step, we give colors to the supplies and demands; each color is correspondent to a client $j \in C$. Notice that $\alpha_v = \sum_{j\in C} x_{U_v,j}$ and $\beta_i = a\sum_{j\in C}\chi_{i,j}$. For every $v \in J, j \in C$, $x_{U_v,j}$ units of demand at v has color j; for every $i \in B, j \in C$, $a\chi_{i,j}$ units of supply at i have color j. In this step, we match the supply and demand using the following greedy rule: while for some $j \in C, i, i' \in B$, there is unmatched demand of color j at v and there is unmatched supply of color j at i, we match them as much as possible. The cost for this step is at most the total cost of moving all supplies and

demands of color j to j, i.e,

$$\begin{aligned}
&\sum_{v\in J, i\in U_v, j\in C} x_{i,j}(d(v,i)+d(i,j)) + a\sum_{i\in B, j\in C}\chi_{i,j}d(i,j)\\
&\le \sum_{v\in J, i\in U_v} x_{i,C}d(v,i) + \sum_{i\in B, j\in C}(x_{i,j}+a\chi_{i,j})d(i,j)\\
&\le O(1)D_B + \sum_{i\in B, j\in C}(x_{i,j}+a\chi_{i,j})d(i,j), \qquad \text{by Lemma 2.3.2.}
\end{aligned}$$

After this step, we have $\sum_{j\in C}\max\{x_{B,j}-a\chi_{B,j},0\} \le \sum_{j\in C}\max\{x_{B,j}-\chi_{B,j},0\}$ units of unmatched demand.

In the second step, we match remaining demand and the supply. For every $v\in J, i\in U_v$, we move the remaining supply at i to v. After this step, all the supplies and the demands are at J; then we match them arbitrarily. The total cost is at most

$$\sum_{v\in J, i\in U_v} a\chi_{i,C}d(i,v) + \sum_{j\in C}\max\{x_{B,j}-\chi_{B,j},0\}\times \mathsf{diam}(J), \tag{2.14}$$

where $\mathsf{diam}(J)$ is the diameter of J.

Notice that $\hat{\mathbb{E}}[\chi_{i,j}] \le 2\ell x_{i,j}$ since $\Pr_{(\chi,\mu)\sim\zeta}[(\chi,\mu)\text{ good}] \ge 1/(2\ell)$ and $\mathbb{E}_{(\chi,\mu)\sim\zeta}\chi_{i,j} = x_{i,j}$. The expected cost of the first step is at most $O(1)D_B + O(\ell)\sum_{j\in C, i\in B} x_{i,j}d(i,j) = O(\ell)D_B$. Similarly, the expected value of the first term of the total cost in (2.14) is at most $O(\ell)\sum_{v\in J, i\in U_v} x_{i,C}d(i,v) \le O(\ell)D_B$ by Lemma 2.3.2.

Consider the second term of (2.14). Notice that $\hat{\mathbb{E}}[\max\{x_{B,j}-\chi_{B,j},0\}] \le x_{B,j}$. Also,

$$\begin{aligned}
\hat{\mathbb{E}}[\max\{x_{B,j}-\chi_{B,j},0\}] &= \hat{\mathbb{E}}[\max\{(1-\chi_{B,j})-(1-x_{B,j}),0\}]\\
&\le \hat{\mathbb{E}}[1-\chi_{B,j}] \le 2\ell(1-x_{B,j}).
\end{aligned}$$

So, $\hat{\mathbb{E}}\max\{x_{B,j}-\chi_{B,j},0\} \le \min\{x_{B,j}, 2\ell(1-x_{B,j})\} \le 3\ell x_{B,j}(1-x_{B,j})$: if $x_{B,j} \ge 1-1/(2\ell) \ge 2/3$, then we have $2\ell(1-x_{B,j}) \le 2\ell(1-x_{B,j})\cdot(3x_{B,j}/2) = 3\ell x_{B,j}(1-x_{B,j})$;

if $x_{B,j} < 1 - 1/(2\ell)$, then $1 - x_{B,j} > 1/(2\ell)$, implying $x_{B,j} \le 2\ell x_{B,j}(1 - x_{B,j})$.

Summing up the inequality over all clients $j \in C$, we have $\hat{\mathbb{E}}\big[\sum_{j\in C} \max\{x_{B,C} - \chi_{B,C}, 0\}\big] \le O(\ell)\pi_J$. So, the expected value of the second term of (2.14) is at most $O(\ell)\pi_J \cdot \mathsf{diam}(J) \le O(\ell^2)\pi_J L(J) \le O(\ell^2) D_B$, by Lemma 2.4.2. This finishes the proof of Lemma 2.4.7. □

At this point, we may have $s_\psi < y_B$. We can apply the following operation repeatedly. Take a pair (S, β) with $\psi_{S,\beta} > 0$ and $S \subsetneq B$. We then shift some ψ-mass from the pair (S, β) to (B, β) so as to increase s_ψ. Thus, we can assume Property (2.4.4a) holds for ψ.

Property (2.4.4d) may be unsatisfied: we only have $\sum_{i\in B} \beta_i \ge x_{B,C}$ for every (S, β) in the support of ψ. To satisfy the property, we focus on each (S, β) in the support of ψ such that $\sum_{i\in B} \beta_i > x_{B,C}$. By considering the matching that achieves $\mathrm{EMD}(\alpha|_J, \beta)$, we can find a $\beta' \in \mathbb{R}^B_{\ge 0}$ such that $\beta'_i \le \beta_i$ for every $i \in B$, $\sum_{i\in B} \beta'_i = \sum_{v\in J} \alpha_v = x_{B,C}$, and $\mathrm{EMD}(\alpha|J, \beta') = \mathrm{EMD}(\alpha|J, \beta)$. We then shift all the ψ-mass at (S, β) to (S, β').

To sum up what we have so far, we have a distribution ψ over (S, β) pairs, that satisfies Properties (2.4.4a), (2.4.4c), (2.4.4d), (2.4.4e) and Property (2.4.4f) with $O(\ell^4)$ replaced with $O(\ell^2)$. The only Property that is missing is Property (2.4.4b); to satisfy the property, we shall apply the following lemma to massage the distribution ψ.

Lemma 2.4.8. *Given a distribution ψ over pairs $(S \subseteq B, \beta \in [0,1]^B)$ satisfying $s_\psi := \mathbb{E}_{(S,\beta)\sim\psi} |S| \le \ell_1$, we can construct another distribution ψ' such that*

(2.4.8a) *$\psi'_{S,\beta} \le O(\ell^2)\psi_{S,\beta}$ for every pair (S, β);*

(2.4.8b) *every pair (S, β) in the support of ψ' has $|S| \le \lceil s_\psi \rceil$;*

(2.4.8c) $\mathbb{E}_{(S,\beta)\sim\psi'} \max\{|S|, \lfloor s_\psi \rfloor\} \le s_\psi$.

Property (2.4.8a) requires that the probability that a pair (S, β) happens in ψ' can not be too large compared to the probability it happens in ψ. Property (2.4.8b) requires $|S| \le \lceil s_\psi \rceil$ for every (S, β) in the support of ψ'. Property (2.4.8c) corresponds to requiring $|S| \ge \lfloor s_\psi \rfloor$:

even if we count the size of S as $\lfloor s_\psi \rfloor$ if $|S| \leq \lfloor s_\psi \rfloor$, the expected size is still going to be at most s_ψ.

Proof of Lemma 2.4.8. If $s_\psi - \lfloor s_\psi \rfloor \leq 1 - 1/\ell$, then we shall throw away the pairs with $|S| > \lfloor s_\psi \rfloor$. More formally, let $Q = \Pr_{(S,\beta)\sim\psi}\left[|S| \leq \lfloor s_\psi \rfloor\right]$ and we define $\psi'_{S,\beta} = \psi_{S,\beta}/Q$ if $|S| \leq \lfloor s_\psi \rfloor$ and $\psi'_{S,\beta} = 0$ if $|S| \geq \lfloor s_\psi \rfloor + 1$. So, Property (2.4.8b) is satisfied. By Markov inequality, we have that $Q \geq 1 - s_\psi/(\lfloor s_\psi \rfloor + 1) = (\lfloor s_\psi \rfloor - s_\psi + 1)/(\lfloor s_\psi \rfloor + 1) \geq (1/\ell)/(\lfloor s_\psi \rfloor + 1) \geq 1/(\ell\ell_1 + \ell)$ since $s_\psi \leq \ell_1 = O(\ell)$. Thus, $\psi'_{S,\beta} \leq O(\ell^2)\psi_{S,\beta}$ for every pair (S,β), implying Property (2.4.8a). Every pair (S,β) in the support of ψ' has $|S| \leq \lfloor s_\psi \rfloor$ and thus Property (2.4.8c) holds.

Now, consider the case where $s_\psi - \lfloor s_\psi \rfloor > 1 - 1/\ell$. In this case, s_ψ is a fractional number. Let ψ'' be the distribution obtained from ψ by conditioning on pairs (S,β) with $|S| \leq \lceil s_\psi \rceil$. By Markov inequality, we have $\Pr_{(S,\beta)\sim\psi}\left[|S| \leq \lceil s_\psi \rceil\right] \geq 1 - s_\psi/(\lceil s_\psi \rceil + 1) \geq 1 - s_\psi/(s_\psi + 1) \geq 1/(\ell_1 + 1)$ as $s_\psi \leq \ell_1 = O(\ell)$. So, $\psi''_{S,\beta} \leq O(\ell)\psi_{S,\beta}$ for every pair (S,β). Moreover, we have $\mathbb{E}_{(S,\beta)\sim\psi''}|S| \leq s_\psi$ since we conditioned on the event that $|S|$ is upper-bounded by some-threshold; all pairs (S,β) in the support of ψ'' have $|S| \leq \lceil s_\psi \rceil$.

Then we modify ψ'' to obtain the final distribution ψ'. Notice that for a pair (S,β) with $|S| \leq \lfloor s_\psi \rfloor$, we have $s_\psi - |S| \leq s_\psi \leq 2\ell_1(s_\psi - \lfloor s_\psi \rfloor)$. Thus,

$$\begin{aligned}\sum_{(S,\beta):|S|\leq\lfloor s_\psi\rfloor} \psi''_{S,\beta}(s_\psi - \lfloor s_\psi \rfloor) &\geq \frac{1}{2\ell_1} \sum_{(S,\beta):|S|\leq\lfloor s_\psi\rfloor} \psi''_{S,\beta}(s_\psi - |S|) \\ &\geq \frac{1}{2\ell_1} \sum_{(S,\beta):|S|=\lceil s_\psi\rceil} \psi''_{S,\beta}(\lceil s_\psi \rceil - s_\psi),\end{aligned}$$

where the second inequality is due to $\mathbb{E}_{(S,\beta)\sim\psi''}|S| \leq s_\psi$.

For every pair (S,β) with $|S| \leq \lfloor s_\psi \rfloor$, let $\psi'_{S,\beta} = \psi''_{S,\beta}$. For every pair (S,β) such that $|S| = \lceil s_\psi \rceil$, we define $\psi'_{S,\beta} = \psi''_{S,\beta}/(2\ell_1)$. Due to the above inequality, we have $\sum_{(S,\beta):|S|\leq\lfloor s_\psi\rfloor} \psi'_{S,\beta}(s_\psi - \lfloor s_\psi \rfloor) \geq \sum_{(S,\beta):|S|=\lceil s_\psi\rceil} \psi'_{S,\beta}(|S| - s_\psi)$ which implies that $\sum_{(S,\beta)} \psi'_{S,\beta} \max\{|S| - s_\psi, \lfloor s_\psi \rfloor - s_\psi\} \leq 0$. Finally, we scale the ψ' vector so that we have

$\sum_{(S,\beta)} \psi'_{S,\beta} = 1$; Properties (2.4.8b) and (2.4.8c) hold. The scaling factor is at most $2\ell_1 = O(\ell)$. Overall, we have $\psi'_{S,\beta} \leq O(\ell^2)\psi_{S,\beta}$ for every pair (S,β) and Property (2.4.8a) holds. This finishes the proof of Lemma 2.4.8. □

With Lemma 2.4.8 we can finish the proof of Theorem 2.4.4 for the case $y_B \leq 2\ell$. We apply the lemma to ψ to obtain the distribution ψ'. By Property (2.4.8a), Properties (2.4.4c), (2.4.4d) and (2.4.4e) remain satisfied for ψ'; Property (2.4.4f) also holds for ψ', as we lost a factor of $O(\ell^2)$ on the expected cost.

To obtain our final distribution ϕ, initially we let $\phi_{S,\beta} = 0$ for every pair (S,β). For every (S,β) in the support of ψ', we apply the following procedure. If $|S| \geq \lfloor s_\psi \rfloor$, then we increase $\phi_{S,\beta}$ by $\psi'_{S,\beta}$; otherwise, take an arbitrary set $S' \subseteq B$ such that $S \subseteq S'$ and $|S'| = \lfloor s_\psi \rfloor$ and increase $\phi_{S',\beta}$ by $\psi_{S,\beta}$. Due to Property (2.4.8b), every pair (S,β) in the support of ϕ has $|S| \in \{\lfloor s_\psi \rfloor, \lceil s_\psi \rceil\}$. Property (2.4.8c) implies that $s_\phi := \mathbb{E}_{(S,\beta)\sim\phi} |S| \leq s_\psi \in [y_B, (1+2\ell\pi)y_B]$. If $s_\phi < s_\psi$, we increase s_ϕ using the following operation. Take an arbitrary pair (S,β) in the support of ϕ such that $|S| = \lfloor s_\psi \rfloor$, let $S' \supseteq S$ be a set such that $S' \subseteq B$ and $|S'| = \lceil s_\psi \rceil$, we decrease $\phi_{S,\beta}$ and increase $\phi_{S',\beta}$. Eventually, we can guarantee $s_\phi = s_\psi$; thus Properties (2.4.4a) and (2.4.4b) are satisfied. This finishes the proof of Theorem 2.4.4 when $y_B \leq 2\ell$.

Now we handle the case where $y_B > 2\ell$. By Properties (2.3.4b) and (2.3.4c), J is a root black component that contains a single representative v and $y_{U_v=B} > 2\ell$. First we find a nearly integral solution with at most $2\ell+2$ open facilities. Then we close two facilities serving the minimum amount of demand and spread their demand among the remaining facilities. Since there is at least 2ℓ open facilities remaining, we increase the amount of demand at any open facility by no more than a factor of $O(1/\ell)$.

Let $u'_i = \frac{x_{i,C}}{y_i} \leq u_i$. We may scale u'_i by a factor of $1 + O(1/\ell)$ during the course of the

algorithm. Consider the following LP with variables $\{\lambda_i\}_{i\in B}$:

$$\min \quad \sum_{i\in U_v} u'_i\lambda_i d(i,v) \quad \text{s.t.} \tag{2.15}$$

$$\sum_{i\in B} u'_i\lambda_i = x_{B,C}; \quad \sum_{i\in B}\lambda_i = y_B; \quad \lambda_i \in [0,1], \quad \forall i \in B.$$

By setting $\lambda_i = y_i$, we obtain a solution to LP(2.15) of value $\sum_{i\in U_v} x_{i,C}d(i,v) \leq O(1)D_{U_v}$, by Lemma 2.3.2. So, the value of LP(2.15) is at most $O(1)D_{U_v}$. Fix on such an optimum vertex-point solution λ of LP(2.15). Since there are only two non-box-constraints, λ has at most two fractional λ_i. Moreover, as $y_{U_v} \geq 2\ell$, there are at least 2ℓ facilities in the support of λ.

We shall reduce the size of the support of λ by 2, by repeating the following procedure twice. Consider the i^* in the support with the smallest $\lambda_{i^*}u'_{i^*}$ value. Let $a := \lambda_{i^*}u'_{i^*}/\sum_{i\in U_v}\lambda_i u'_i \leq O(\frac{1}{\ell})$, we then scale u'_i by a factor of $1/(1-a) \leq 1 + O(1/\ell)$ for every $i \in U_v \setminus i^*$ and change λ_{i^*} to 0. So, we still have $\sum_{i\in U_v} u'_i\lambda_i = x_{U_v,C}$. The value of the objective function is scaled by a factor of at most $1 + O(1/\ell)$.

Let $S = \{i \in U_v : \lambda_i > 0\}$ and let $\beta_i = \lambda_i u'_i$ for every $i \in U_v$. So, $|S| \leq y_{U_v}$ Properties (2.4.4c) and (2.4.4d) are satisfied. Moreover, $\mathrm{EMD}_J(\alpha|_J, \beta) \leq \sum_{i\in U_v}\beta_i d(i,v) \leq O(1)D_{U_v}$ since the value of LP(2.15) is $O(1)D_{U_v}$ and we have scaled each β_i by at most a factor of $1 + O(1/\ell)$.

If we let ϕ contains the single pair (S,β) with probability 1, then all properties from (2.4.4c) to (2.4.4f) are satisfied. To satisfy Properties (2.4.4a) and (2.4.4b), we can manually add facilities to S with some probability, as we did before for the case $y_B \leq 2\ell$. This finishes the proof of Theorem 2.4.4.

2.4.3 Local Solutions for Unions of Non-Concentrated Components

In this section, we construct a local solution for the union V of some non-concentrated black components that are close to each other.

Lemma 2.4.9. *Let $\mathcal{J}' \subseteq \mathcal{J}^{\mathsf{N}}$ be a set of non-concentrated black components, $V = \bigcup_{J\in\mathcal{J}'} J$ and $B = U(V)$. Assume there exists $v^* \in R$ such that $d(v, v^*) \le O(\ell^2)L(J)$ for every $J \in \mathcal{J}'$ and $v \in J$. Then, we can find a pair $(S \subseteq B, \beta \subseteq \mathbb{R}^B_{\ge 0})$ such that*

(2.4.9a) $|S| \in \left\{ \lceil y_B \rceil, \lceil y_B \rceil + 1 \right\}$;

(2.4.9b) $\beta_i \le u_i$ *if* $i \in S$ *and* $\beta_i = 0$ *if* $i \in B \setminus S$;

(2.4.9c) $\sum_{i\in S} \beta_i = x_{B,C} = \sum_{v\in V} \alpha_v$;

(2.4.9d) $\mathrm{EMD}_V(\alpha|_V, \beta) \le O(\ell^2\ell_2)D_B$.

Proof. We shall use an algorithm similar to the one we used for handling the case where $y_B > 2\ell$ in Section 2.4.2. Again, for simplicity, we let $u'_i = \frac{x_{i,C}}{y_i} \le u_i$ to be the "effective capacity" of i. Consider the following LP with variables $\{\lambda_i\}_{i\in B}$:

$$\min \sum_{J\in\mathcal{J}', v\in J, i\in U_v} u'_i\lambda_i\big(d(i,v) + \ell^2 L(J)\big) \quad \text{s.t.} \tag{2.16}$$

$$\sum_{i\in B} u'_i\lambda_i = x_{B,C}; \quad \sum_{i\in B} \lambda_i = y_B; \quad \lambda_i \in [0,1], \quad \forall i \in B.$$

By setting $\lambda_i = y_i$, we obtain a valid solution to the LP with the objective value

$$\begin{aligned}
&\sum_{J\in\mathcal{J}', v\in J, i\in U_v} x_{i,C}\big(d(i,v) + \ell^2 L(J)\big) \\
&\le \sum_{v\in V, i\in U_v} x_{i,C} d(i,v) + \ell^2 \sum_{J\in\mathcal{J}'} x_{U(J),C} L(J) \le \sum_{v\in V} O(1) D_{U_v} + \ell^2 \sum_{J\in\mathcal{J}'} \ell_2 \pi_J L(J) \\
&\le O(1) D_B + \ell^2\ell_2 \sum_{J\in\mathcal{J}'} O(1) D_{U(J)} = O(\ell^2\ell_2) D_B,
\end{aligned} \tag{2.17}$$

by Lemma 2.3.2 and Lemma 2.4.2. So, the value of LP(2.16) is at most $O(\ell^2\ell_2)D_B$.

Fix such an optimum vertex-point solution λ of LP (2.16). Since there are only two non-box-constraints, every vertex-point λ of the polytope has at most two fractional λ_i.

Let $S = \{i \in B : \lambda_i > 0\}$ and let $\beta_i = \lambda_i u'_i$ for every $i \in B$. So, Properties (2.4.9a), (2.4.9b) and (2.4.9c) are satisfied.

Now we prove Property (2.4.9d). To compute $\mathrm{EMD}_V(\alpha|_V, \beta)$, we move all demands in $\alpha|_V$ and all supplies in β to v^*. The cost is

$$\begin{aligned}
&\sum_{v\in V} \alpha_v d(v, v^*) + \sum_{i\in B} \beta_i d(i, v^*) \\
&\leq \sum_{J\in\mathcal{J}', v\in J} x_{U_v,C} O(\ell^2) L(J) + \sum_{J\in\mathcal{J}', v\in J, i\in U_v} \beta_i \big(d(i, v) + O(\ell^2)L(J)\big) \leq O(\ell^2\ell_2)D_B,
\end{aligned}$$

where the $O(\ell^2\ell_2)D_B$ for the first term was proved in (2.17) and the bound $O(\ell^2\ell_2)D_B$ for the second term is due to the fact that γ is an optimum solution to LP(2.16). This finishes the proof of Lemma 2.4.9. □

2.5 Rounding Algorithm

In this section we describe our rounding algorithm. We start by giving the intuition behind the algorithm. For each concentrated component $J \in \mathcal{J}$, we construct a distribution of local solutions using Theorem 2.4.4. We shall construct a partition $\mathcal{V}^{\mathsf{N}}$ of the representatives in $\bigcup_{J\in\mathcal{J}^{\mathsf{N}}} J$ so that each $V \in \mathcal{V}^{\mathsf{N}}$ is the union of some nearby components in $\mathcal{J}^{\mathsf{N}}$. For each set $V \in \mathcal{V}^{\mathsf{N}}$, we apply Lemma 2.4.9 to construct a local solution. If we independently and randomly choose a local solution from every distribution we constructed, then we can move all the demands to the open facilities at a small cost, by Property (2.4.4f) and Property (2.4.9d).

However, we may open more than k facilities, even in expectation. Noticing that the fractional solution opens y_B facilities in a set B, the extra number of facilities come from two

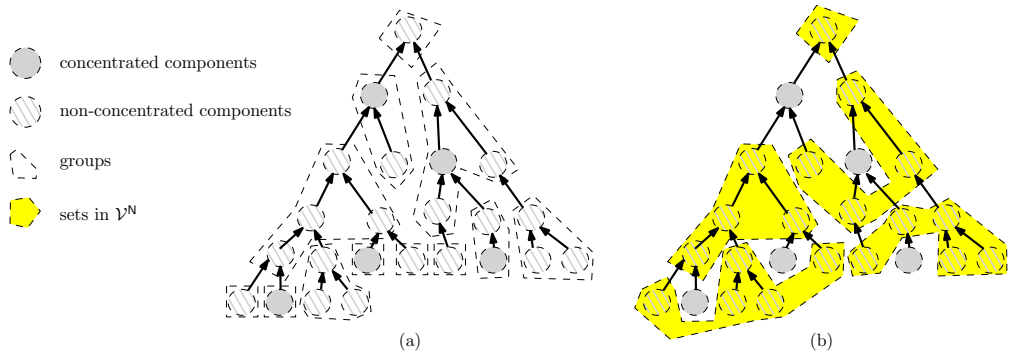


Figure 2.2: Figure (a) gives the forest $\Upsilon^*_{\mathcal{J}}$ over $\mathcal{J}$ and the set $\mathcal{G}$ of groups (denoted by empty polygons). Figure (b) gives $\mathcal{V}^{\mathsf{N}}$: each set $V \in \mathcal{V}^{\mathsf{N}}$ is the union of components in a solid polygon.

places. In Property (2.4.4a) of Theorem 2.4.4, we may open in expectation $y_B \cdot 2\ell\pi_J/x_{B,C}$ more facilities in B than y_B. Then in Property (2.4.9a) of Lemma 2.4.9, we may open $\lceil y_B \rceil$ or $\lceil y_B \rceil + 1$ facilities in B. To reduce the number of open facilities to k, we shall shut down (or remove) some already-open facilities and move the demands satisfied by these facilities to the survived open facilities: a concentrated component $J \in \mathcal{J}^{\mathsf{C}}$ is responsible for removing $y_B \cdot 2\ell\pi_J/x_{B,C} < 1$ facilities in expectation; a set $V \in \mathcal{V}^{\mathsf{N}}$ is responsible for removing up to 2 facilities. Lemma 2.4.2 allows us to bound the cost of moving demands caused by the removal, provided that the moving distance is not too big. To respect the capacity constraint up to a factor of $1+\epsilon$, we are only allowed to scale the supplies of the survived open facilities by a factor of $1 + O(1/\ell)$. Both requirements will be satisfied by the forest structure over groups and the fact that each non-leaf group contains $\Omega(\ell)$ fractional opening (Property (2.3.5c)). Due to the forest structure and Property (2.3.5c), we always have enough open facilities locally that can support the removing of facilities.

In order to guarantee that we always open k facilities, we need to use a dependent rounding procedure for opening and removing facilities. As in many of previous algorithms, we incorporate the randomized rounding procedure into random selections of vertex points of polytopes respecting marginal probabilities. In many cases, a randomized selection procedure can be derandomized since there is an explicit linear objective we shall optimize.

We now formally describe our rounding algorithm. For every group $G \in \mathcal{G}$, we use Λ_G to denote the set of child-groups of G. We construct a partition $\mathbb{J}^{\mathsf{C}}$ of $\mathcal{J}^{\mathsf{C}}$ as follows. For each root group $G \in \mathcal{G}$, we add $G \cap \mathcal{J}^{\mathsf{C}}$ to $\mathbb{J}^{\mathsf{C}}$ if it is not empty. For each non-leaf group $G \in \mathcal{G}$, we add $\bigcup_{G' \in \Lambda_G}(G' \cap \mathcal{J}^{\mathsf{C}})$ to $\mathbb{J}^{\mathsf{C}}$, if it is not empty. We construct the partition $\mathbb{J}^{\mathsf{N}}$ for $\mathcal{J}^{\mathsf{N}}$ in the same way, except that we consider components in $\mathcal{J}^{\mathsf{N}}$. We also define a set $\mathcal{V}^{\mathsf{N}}$ as follows: for every $\mathcal{J}' \in \mathbb{J}^{\mathsf{N}}$, we add $\bigcup_{J \in \mathcal{J}'} J$ to $\mathcal{V}^{\mathsf{N}}$; thus, $\mathcal{V}^{\mathsf{N}}$ forms a partition for $\bigcup_{J \in \mathcal{J}^{\mathsf{N}}} J$. See Figure 2.2 for the definition of $\mathcal{V}^{\mathsf{N}}$.

In Section 2.5.1, we describe the procedure for opening a set S^* of facilities, whose cardinality may be larger than k. Then in Section 2.5.2, we define the procedure remove, which removes one open facility. We wrap up the algorithm in Section 2.5.3.

2.5.1 Constructing Initial Set S^* of Open Facilities

In this section, we open a set S^* of facilities, whose cardinality may be larger than k, and construct a supply vector $\beta^* \in \mathbb{R}^F_{\geq 0}$ such that $\beta^*_i = 0$ if $i \notin S^*$. (S^*, β^*) will be the concatenation of all local solutions we constructed.

It is easy to construct local solutions for non-concentrated components. For each set $\mathcal{J}' \in \mathbb{J}^{\mathsf{N}}$ of components and its correspondent $V = \bigcup_{J \in \mathcal{J}'} J \in \mathcal{V}^{\mathsf{N}}$, we apply Lemma 2.4.9 to obtain a local solution $\left(S \subseteq U(V), \beta \in \mathbb{R}^{U(V)}_{\geq 0}\right)$. Then, we add S to S^* and let $\beta^*_i = \beta_i$ for every $i \in U(V)$. Notice that $\mathcal{J}'$ either contains a single root black component J, or contains all the non-concentrated black components in the child-groups of some group G. In the former case, the diameter of J is at most $O(\ell)L(J)$ by Property (2.3.4a); in the latter case, we let v^* be an arbitrary representative in $\bigcup_{J' \in G} J'$ and then any representative $v \in J, J \in \mathcal{J}'$ has $d(v, v^*) \leq O(\ell^2)L(J)$ by Property (2.3.5d). Thus, all the properties in Lemma 2.4.9 are satisfied.

For concentrated components, we only obtain distributions of local solutions by applying Theorem 2.4.4. For every $J \in \mathcal{J}^{\mathsf{C}}$, we check if Constraints (2.6) to (2.12) are satisfied for $B = U(J)$. If not, we return a separation plane for the fractional solution; otherwise we

apply Theorem 2.4.4 to each component J to obtain a distribution $\left(\phi^J_{S,\beta}\right)_{S\subseteq U(J),\beta\in\mathbb{R}^{U(J)}_{\geq 0}}$. To produce local solutions for concentrated components, we shall use a dependent rounding procedure that respects the marginal probabilities. As mentioned earlier, we shall define a polytope and the procedure randomly selects a vertex point of the polytope.

We let $s_J := s_{\phi_J} := \mathbb{E}_{(S,\beta)\sim\phi^J}|S|$ be the expectation of $|S|$ according to distribution ϕ^J. For notational convenience, we shall use $a \approx b$ to denote $a \in \left[\lfloor b\rfloor, \lceil b\rceil\right]$. Consider the following polytope $\mathcal{P}$ defined by variables $\{\psi^J_{S,\beta}\}_{J\in\mathcal{J}^{\mathsf{C}},S,\beta}$ and $\{q_J\}_{J\in\mathcal{J}^{\mathsf{C}}}$.[3]

$$\psi^J_{S,\beta}, p_J \in [0,1]\ \forall J\in\mathcal{J}^{\mathsf{C}}, S, \beta; \quad (2.18) \qquad \sum_{J\in\mathcal{J}'} q_J \leq 1, \qquad \forall \mathcal{J}'\in\mathbb{J}^{\mathsf{C}}; \quad (2.20)$$

$$\sum_{S,\beta}\psi^J_{S,\beta} = 1, \quad \forall J\in\mathcal{J}^{\mathsf{C}}; \quad (2.19) \qquad \sum_{S,\beta}\psi^J_{S,\beta}|S| - q_J \approx y_{U(J)}, \quad \forall J\in\mathcal{J}^{\mathsf{C}}; \quad (2.21)$$

$$\sum_{J\in\mathcal{J}'}\Big(\sum_{S,\beta}\psi^J_{S,\beta}|S| - q_J\Big) \approx \sum_{J\in\mathcal{J}'} y_{U(J)}, \quad \forall \mathcal{J}'\in\mathbb{J}^{\mathsf{C}}; \quad (2.22)$$

$$\sum_{J\in\mathcal{J}^{\mathsf{C}}}\Big(\sum_{S,\beta}\psi^J_{S,\beta}|S| - q_J\Big) \approx \sum_{J\in\mathcal{J}^{\mathsf{C}}} y_{U(J)}. \quad (2.23)$$

In the above LP, ψ^J is the indicator vector for local solutions for J and q_J indicates whether J is responsible for removing one facility; if $q_J = 1$, we shall call $\mathsf{remove}(J)$ later. Up to changing of variables, any vertex point of $\mathcal{P}$ is defined by two laminar families of tight constraints and thus $\mathcal{P}$ is integral:

Lemma 2.5.1. *$\mathcal{P}$ is integral.*

Proof. To avoid negative coefficients, we shall let $q'_J = 1 - q_J$ and focus on ψ and q' variables. Consider the set of tight constraints that define a vertex point. The tight constraints from (2.18), (2.19) and (2.20) define a matroid base polytope for a laminar matroid.

For each $J\in\mathcal{J}$, every pair (S,β) in the support of ϕ^J has $|S| \approx s_J$. Thus, Con-

3. For every $J\in\mathcal{J}^{\mathsf{C}}$, we only consider the pairs (S,β) in the support of ϕ^J; thus the total number of variables is $n^{O(\ell)}$.

straint (2.21) is equivalent to $\sum_{S,\beta} \psi^J_{S,\beta}(|S| - \lfloor s_J \rfloor) + q'_J \approx y_{U(J)} + 1 - \lfloor s_J \rfloor$. This is true since Constraint (2.19) holds and ψ^J is a distribution. We do the same transformation for Constraints (2.22) and (2.23). It is easy to see that the tight constraints from (2.21), (2.22) and (2.23) also define the matroid-base-polytope for a laminar matroid.

Thus, by the classic matroid theory, the set of tight constraints define an integral solution; thus $\mathcal{P}$ is integral. □

We set $\psi^{*J}_{S,\beta} = \phi^J_{S,\beta}$ and $q^*_J = s_J - y_{U(J)}$ for every $J \in \mathcal{J}^{\mathsf{C}}$ and (S, β). Then,

Lemma 2.5.2. *(ψ^*, q^*) is a point in polytope $\mathcal{P}$.*

Proof. Notice that s_J is the expected size of S according to the distribution ϕ^J, while $y_{U(J)}$ is the budget for the number of open facilities open in $U(J)$. So q^*_J is the expected number of facilities that go beyond the budget. It is easy to see that Constraints (2.18) and (2.19) hold for ψ^* and q^*. For every $\mathcal{J}' \in \mathbb{J}^{\mathsf{C}}$, we have that $\sum_{J\in\mathcal{J}'} q^*_J \le 2\ell \sum_{J\in\mathcal{J}'} y_{U(J)}\pi_J / x_{U(J),C} \le 2\ell \sum_{J\in\mathcal{J}'} y_{U(J)}/\ell_2 \le 2\ell \times O(\ell^2)/\ell_2$ due to Properties (2.3.5e) and (2.4.4a). This at most 1 if $\ell_2 = \Theta(\ell^3)$ is large enough. (If $\mathcal{J}'$ contains a root component J which has $y_{U(J)} > 2\ell$ then $\sum_{J\in\mathcal{J}'} q^*_J = 0$.) Thus, Constraint (2.20) holds. $\sum_{S,\beta} \psi^{*J}_{S,\beta}|S| - q^*_J = s_J - q^*_J = y_{U(J)}$. So, Constraints (2.21), (2.22) and (2.23) hold. So, (ψ^*, p^*) is a point in $\mathcal{P}$. □

We randomly select a vertex point (ψ, q) of $\mathcal{P}$ such that $\mathbb{E}[\psi^J_{S,\beta}] = \psi^{*J}_{S,\beta} = \phi^J_{S,\beta}$ for every $J \in \mathcal{J}^{\mathsf{C}}, (S, \beta)$, and $\mathbb{E}[q_J] = q^*_J = s_J - y_{U(J)}$ for every $J \in \mathcal{J}^{\mathsf{C}}$. Since ψ is integral, for every $J \in \mathcal{J}$, there is a unique local solution $\left(S \subseteq U(J), \beta \in \mathbb{R}^{U(J)}_{\ge 0}\right)$ such that $\psi^J_{S,\beta} = 1$; we add S to S^* and let $\beta^*_i = \beta_i$ for every $i \in U(J)$.

This finishes the definition of the initial S^* and β^*. Let $\alpha^* = \alpha$ (recall that $\alpha_v = x_{U_v,C}$ is the demand at v after the initial moving, for every $v \in R$) be the initial demand vector. Later we shall remove facilities from S^* and update α^* and β^*. S^*, α^*, β^* satisfy the following properties, which will be maintained as the rounding algorithm proceeds.

(2.5.3a) $\sum_{v\in V} \alpha^*_v = \sum_{v\in V} \beta^*_v$ for every $V \in \mathcal{J}^{\mathsf{C}} \cup \mathcal{V}^{\mathsf{N}}$;

(2.5.3b) $\sum_{v\in R} \alpha^*_v = |C|$.

Property (2.5.3a) is due to Properties (2.4.4d) and (2.4.9c). Property (2.5.3b) holds since $\sum_{v\in R}\alpha^*_v = \sum_{v\in R} x_{U_v,C} = x_{F,C} = |C|$.

2.5.2 The remove *procedure*

In this section, we define the procedure remove that removes facilities from S^* and updates α^* and β^*. The procedure takes a set $V \in \mathcal{J}^{\mathsf{C}} \cup \mathcal{V}^{\mathsf{N}}$ as input. If V is a root black component, then we let $G = \{V\}$ be the root group containing V; if V is a non-root concentrated component, let G be the parent group of the group containing V; otherwise V is the union of non-concentrated components in all child-groups of some group, and we let G be this group. Let $V' = \bigcup_{J'\in G} J'$. Before calling remove$(V)$, we require the following properties to hold:

(2.5.4a) $|S^* \cap U(V)| \geq 1$;

(2.5.4b) $\left|S^* \cap U(V')\right| \geq \ell - 6$.

While maintaining Properties (2.5.3a) and (2.5.3b), the procedure remove(V) will

(2.5.5a) remove from S^* exactly one open facility, which is in $U(V \cup V')$,

(2.5.5b) not change $\alpha^*|_{R\setminus(V\cup V')}$ and $\beta^*|_{F\setminus U(V\cup V')}$,

(2.5.5c) increase α^*_v by at most a factor of $1 + O(1/\ell)$ for every $v \in V \cup V'$ and increase β^*_i by at most a factor of $1 + O(1/\ell)$ for every $i \in U(V \cup V')$.

Moreover,

(2.5.5d) the moving cost for converting the old α^* to the new α^* is at most $O(\ell^2)\beta^*_{i^*}L(J)$ for some black component $J \subseteq V$ and facility $i^* \in U(J)$;

(2.5.5e) for every $V'' \in \mathcal{J}^{\mathsf{C}} \cup \mathcal{V}^{\mathsf{N}}$, $\mathrm{EMD}_{V''}\big(\alpha^*|_{V''}, \beta^*|_{U(V'')}\big)$ will be increased by at most a factor of $1 + O(1/\ell)$.

Before formally describe the procedure remove(V), we first highlight some key ideas. Assume V is not a root component. We choose an arbitrary facility $i \in S^* \cap U(V)$. Notice that there are $\Omega(\ell)$ facilities in $S^* \cap U(V')$. If the $\beta^*_i \leq \sum_{v'\in V'} \alpha^*_{v'}/\ell$, then we can shut down

i and send the demands that should be sent to i to V'. We only need to increase the supplies in $U(V')$ by a factor of $1+O(1/\ell)$. Otherwise, we shall shut down the facility $i' \in S^* \cap U(V')$ with the smallest $\beta^*_{i'}$ value. Since there are at least $\Omega(\ell)$ facilities in $U(V')$, we can satisfy the $\beta^*_{i'}$ units of unsatisfied demands using other facilities in $S^* \cap U(V')$. For this i', we have $\beta^*_{i'} \leq O(1)\beta^*_i$. Thus, the total amount of demands that will be moved is comparable to β^*_i. In either case, the cost of redistributing the demands is not too big. When V is a root component, we shall shut down the facility $i' \in S^* \cap U(V)$ with the smallest $\beta^*_{i'}$ value.

We now formally describe the procedure $\mathsf{remove}(V)$. We first consider the case that V is not a root component. So, V is either a non-root component in $\mathcal{J}^{\mathsf{C}}$, or the union of all non-concentrated components in all child-groups of the group G. In this case, $V \cap V' = \emptyset$. Let $i \in U(V) \cap S^*$ be an arbitrary facility; due to Property (2.5.4a), we can find such an i. Let $J \subseteq V$ be the component that contains i.

If $a := \beta^*_i / \sum_{v' \in V'} \alpha^*_{v'} \leq \frac{1}{\ell}$, then we shall shutdown i. Consider the matching $f : V \times U(V) \to \mathbb{R}_{\geq 0}$ between $\alpha^*|_V$ and $\beta^*|_{U(V)}$ that achieves $\mathrm{EMD}_V\big(\alpha^*|_V, \beta^*|_{U(V)}\big)$. (Due to Property (2.5.3a), the total supply equals the total demand.) For every $v \in V$, we shall move $f(v,i)$ units of demand from v to V'. The total amount of demand moved from V to V' is exactly $\sum_{v \in V} f_{v,i} = \beta^*_i$. Every $v' \in V'$ will receive $a\alpha^*_{v'}$ units of demand. We update α^* to be the new demand vector: decrease α^*_v by $f(v,i)$ for every $v \in V$ and scale $\alpha^*_{v'}$ by a factor of $(1+a)$ for every $v' \in V'$. By Property (2.3.5d), the cost of moving the demands is at most $\beta^*_i O(\ell^2) L(J)$; thus, Property (2.5.5d) holds.

We remove i from S^*, change β^*_i to 0, and for every $i' \in U(V')$, scale $\beta^*_{i'}$ by a factor of $(1+a)$. For this new α^* and β^* vector, $\mathrm{EMD}_V(\alpha^*, \beta^*)$ will not increase. $\alpha^*|_{V''}$ and $\beta^*|_{U(V'')}$ are scaled by a factor of $1 + a \leq 1 + 1/\ell$ for every $V'' \in \mathcal{J}^{\mathsf{C}} \cup \mathcal{V}^{\mathsf{N}}$ such that $V'' \subseteq V'$. Thus Properties (2.5.5c) and (2.5.5e) are satisfied. Properties (2.5.5a) and (2.5.5b) are trivially true. Moreover, we maintained Properties (2.5.3a) and (2.5.3b).

Now consider the case $a > 1/\ell$. In this case, we shall remove the facility $i' \in S^* \cap U(V')$ with the smallest $\beta^*_{i'}$ value from S^*. Notice that we have $|S^* \cap U(V')| \geq \ell - 6$ before we run

$\mathsf{remove}(V)$ due to Property (2.5.4b). Let $a' := \beta_{i'} / \sum_{i'' \in U(V')} \beta^*_{i''}$; so, we have $a' \leq 1/(\ell - 6)$. To remove the facility i', we consider the function f that achieves $\mathrm{EMD}_{V'}(\alpha^*|_{V'}, \beta^*|_{U(V')})$. We shall redistribute the demands in V' so that the new demand at $v' \in V'$ will be $(1 + a') \sum_{i'' \in U(V') \setminus \{i'\}} f(v', i'')$. We remove i' from S^*, change $\beta^*_{i'}$ to 0 and scale up $\beta^*_{i''}$ for all other $i'' \in U(V') \setminus \{i'\}$ by $(1 + a')$. Then, the total cost for redistributing the demands in this procedure will be at most $\beta^*_{i'} O(\ell) L(J)$, due to Property (2.3.4d). This is at most $O(\ell)\beta^*_i L(J)$ since $a > 1/\ell$ and $a' \leq 1/(\ell - 6)$. So, Properties (2.5.5a) to (2.5.5e) are satisfied and Properties (2.5.3a) and (2.5.3b) are maintained.

The case where V is a root component can be handled in a similar way. In this case, we have $G = \{V\}$ and $V' = V$. By Property (2.5.4b), there are at least $\ell - 6$ facilities in $S^* \cap U(V)$. Then we can remove the facility $i \in U(V) \cap S^*$ with the smallest β^*_i. Using the same argument as above, we can guarantee Properties (2.5.5a) to (2.5.5e) and maintain Properties (2.5.3a) and (2.5.3b).

2.5.3 Obtaining the Final Solution

To obtain our final set S^* of facilities, we call the remove procedures in some order. We consider each group G using the top-to-bottom order. That is, before we consider a group G, we have already considered its parent group. If G is a root group, then it contains a single root component J. If $J \in \mathcal{J}^{\mathsf{N}}$, repeat the the following procedure twice: if there is some facility in $S^* \cap U(J)$ then we call $\mathsf{remove}(J)$. If $J \in \mathcal{J}^{\mathsf{C}}$ and $q_J = 1$ then we call $\mathsf{remove}(J)$. Now if G is a non-leaf group, then do the following. Let $V = \bigcup_{G' \in \Lambda_G, J \in G' \cap \mathcal{J}^{\mathsf{N}}} J$. Repeat the following procedure twice: if there is some facility in $S^* \cap U(V)$ then we call $\mathsf{remove}(V)$. For every $G' \in \Lambda_G$ and $J \in G' \cap \mathcal{J}^{\mathsf{C}}$ such that $q_J = 1$ we call $\mathsf{remove}(J)$.

Lemma 2.5.6. *After the above procedure, we have $|S^*| \leq y_F \leq k$.*

Proof. We first show that whenever we call $\mathsf{remove}(V)$, Properties (2.5.4a) and (2.5.4b) hold. For any concentrated component J with $q_J = 1$, we have called $\mathsf{remove}(J)$. Notice that if

$q_J = 1$, then initially we have $|S^* \cap U(J)| \geq 1$ due to Constraint (2.21). Due to the top-down order of considering components, and Property (2.5.5a), we have never removed a facility in $S^* \cap U(J)$ before calling $\mathsf{remove}(J)$. Thus, Property (2.5.4a) holds. For $V \in \mathcal{V}^{\mathsf{N}}$, we check if $|S^* \cap U(V)| \geq 1$ before we call $\mathsf{remove}(V)$ and thus Property (2.5.4a) holds.

Now consider Property (2.5.4b). For any non-leaf group G, initially, we have $\left|S^* \cap \bigcup_{J \in G} U(J)\right| \geq \left\lfloor \sum_{J \in G} y_{U(J)} \right\rfloor \geq \ell$ where the first inequality is due to Property (2.4.9a) and Constraint (2.22) and the second is due to Property (2.3.5c). We may remove a facility from the set when we call $\mathsf{remove}(V)$ for V satisfying one of the following conditions: (a) V is a concentrated component in G or in a child group of G, (b) V is the union of the non-concentrated components in the child-groups of G or (c) V contains the non-concentrated components in G. For case (a), we removed at most 2 facilities due to Constraint (2.20). For each (b) and (c), we remove at most 2 facilities. Thus, we shall remove at most 6 facilities from $\left|S^* \cap \bigcup_{J \in G} U(J)\right| \geq \sum_{J \in G} y_{U(J)}$. Thus, Property (2.5.4b) holds.

Thus, every call of remove is successful. For a concentrated component J with $q_J = 1$, we called $\mathsf{remove}(J)$ once. For each $V \in \mathcal{V}^{\mathsf{N}}$, initially we have $|S^* \cap U(V)| \in \left\{ \left\lceil y_{U(V)} \right\rceil, \left\lceil y_{U(V)} \right\rceil + 1 \right\}$. Before calling $\mathsf{remove}(V)$, we have never removed a facility from $S^* \cap U(V)$. Thus, the number of times we call $\mathsf{remove}(V)$ is at least the initial value of $|S^* \cap U(V)|$ minus $y_{U(V)}$. Overall, the number of facilities in S^* after the removing procedure is at most $\sum_{J \in \mathcal{J}^{\mathsf{C}}} \left(\sum_{S,\beta} \psi^J_{S,\beta} - q_J \right) + \sum_{V \in \mathcal{V}^{\mathsf{N}}} y_{U(V)} < \sum_{J \in \mathcal{J}^{\mathsf{C}}} y_{U(J)} + 1 + \sum_{V \in \mathcal{V}^{\mathsf{N}}} y_{U(V)} = y_F + 1 \leq k + 1$, where the first inequality is due to Constraint (2.23). Since $|S^*|$ is an integer, we have that $|S^*| \leq k$. □

By Properties (2.5.5b) and (2.5.5c), and Constraint (2.20), our final β^*_i is at most $1 + O(1/\ell)$ times the initial β^*_i for every $i \in V$. Finally we have $\beta^*_i \leq (1 + O(1/\ell))u_i$ for every $i \in F$. Thus, the capacity constraint is violated by a factor of $1 + \epsilon$ if we set ℓ to be large enough.

It remains to bound the expected cost of the solution S^*; this is done by bounding the cost for transferring the original α^* to the final α^*, as well as the cost for matching our final

α^* and β^*.

We first focus on the transferring cost. By Property (2.5.5e), when we call $\mathsf{remove}(V)$, the transferring cost is at most $O(\ell^2)\beta^*_{i^*}L(J)$ for some black component $J \subseteq V$ and i^*. Notice that $\beta^*_{i^*}$ is scaled by at most a factor of $(1 + O(1/\ell))$, we always have $\beta^*_{i^*} \leq (1+O(1/\ell))\alpha_{U(J),C}$. So, the cost is at most $O(\ell^2)x_{U(J),C}L(J)$. If V is the union of some non-concentrated components, then this quantity is at most $O(\ell^2)\ell_2\pi_J L(J) \leq O(\ell^2\ell_2)D_{U(J)} \leq O(\ell^2\ell_2)D_{U(V)}$. We call $\mathsf{remove}(V)$ at most twice, thus the contribution of V to the transferring cost is at most $O(\ell^2\ell_2)D_{U(V)}$. If V is a concentrated component J, then the quantity might be large. However, the probability we call $\mathsf{remove}(J)$ is $\mathbb{E}[q_J] = q^*_J = s_J - y_{U(J)} \leq 2\ell y_{U(J)}\pi_J/x_{U(J),C}$ if $y_{U(J)} \leq 2\ell$ and it is 0 otherwise (by Property (2.4.4a)). So, the expected contribution of this V to the transferring cost is at most $O(\ell^2)x_{U(J),C}L(J) \times 2\ell y_{U(J)}\pi_J/x_{U(J),C} \leq O(\ell^4)\pi_J L(J) \leq O(\ell^4)D_{U(J)}$ by Lemma 2.4.2. Thus, overall, the expected transferring cost is at most $O(\ell^5)D_F = O(\ell^5)\mathsf{LP}$.

Then we consider the matching cost. Since we maintained Property (2.5.3a), the matching cost is bounded by $\sum_{V\in\mathcal{J}^{\mathsf{C}}\cup\mathcal{V}^{\mathsf{N}}} \mathrm{EMD}_V(\alpha^*|_V, \beta^*|_{U(V)})$. Due to Property (2.5.5e), this quantity has only increased by a factor of $1+O(1/\ell)$ during the course of removing facilities. For the initial α^* and β^*, the expectation of this quantity is at most $\sum_{J\in\mathcal{J}^{\mathsf{C}}} O(\ell^4)D_{U(J)} + \sum_{V\in\mathcal{V}^{\mathsf{N}}} O(\ell^2\ell_2)D_{U(V)}$ due to Properties (2.4.4f) and (2.4.9d). This is at most $O(\ell^5)D_F = O(\ell^5)\mathsf{LP}$.

We have found a set S^* of at most k facilities and a vector $\beta^* \in \mathbb{R}^F_{\geq 0}$ such that $\beta^*_i = 0$ for every $i \notin S^*$ and $\beta^*_i \leq (1 + O(1/\ell))u_i$. If we set $\ell = \Theta(1/\epsilon)$ to be large enough, then $\beta^*_i \leq (1 + \epsilon)u_i$. The cost for matching the α-demand vector and the β^* vector is at most $O(\ell^5)\mathsf{LP} = O(1/\epsilon^5)\mathsf{LP}$. Thus, we obtained a $O(1/\epsilon^5)$-approximation for CKM with $(1 + \epsilon)$-capacity violation.

CHAPTER 3
SCHEDULING WITH RESOURCE AND PRECEDENCE CONSTRAINTS

3.1 Introduction

Scheduling under precedence constraints and scheduling under resource constraints are both well studied topics in scheduling theory and approximation algorithms. In the former, a precedence relationship between jobs is given; if a job j has precedence over another, say j', then j must be completed before j' can start. In the latter, each job has a resource requirement and there is a finite amount of resource; when jobs run in parallel their total resource requirement must not exceed the given resource capacity. Both of these problems are central to scheduling theory, and their approximability is well understood. They have $(2 - 1/m)$- and $(2+\epsilon)$- approximation algorithms, respectively, where m is the number of identical machines. However, the approximability of the natural problem with the combination of these constraints is still wide open. We study this problem, namely scheduling on parallel machines under both resource and precedence constraints, and give the first non-trivial approximation algorithms for several important versions of this problem.

The combination of these two constraints naturally models the computation in emerging high performance computing systems. Power consumption is one of the central design considerations for the next generation of *exascale* supercomputers [80]. These future systems will have to run parallel computations within a 20 MW operating budget, but will be built such that if all processors were used at full capacity the budget would be exceeded and the system would fail catastrophically [17]. Therefore, exascale system software must schedule parallel computations (with precedence constraints) on this hardware while respecting the global power budget (a limited shared resource) and minimizing application runtime [88]. This critical problem for emerging supercomputers is a practical example of a scheduling with simultaneous resource and precedence constraints. In case of this motivating problem,

the resource is power and the resource cap is the power budget allocated to the entire system.

3.1.1 Problem Definition

We now define the problem formally. We are given a set $\mathcal{J}$ of n jobs to be scheduled on m parallel machines. The schedule needs to be non-preemptive; i.e. each job, once assigned to a machine at some point in time, must run to completion on the same machine without interruption. Each job $j \in \mathcal{J}$ has a processing time $p_j \in \mathbb{Z}_{\geq 0}$ and a resource requirement $s_j \in \mathbb{Z}_{\geq 0}$. There is a global resource capacity $S \in \mathbb{Z}_{\geq 0}$ which any feasible schedule must respect. That is, the sum of the resource requirements of the jobs running on the m machines at any point in time must be at most S. Moreover, we have precedence constraints given by a partial order $\prec$ on the jobs such that if $j \prec j'$ then j must complete before j' can start.

We study both *identical* machines and *uniformly related* machines. The difference between the two is the time it takes to process a job—on identical machines, it takes p_j units of time to complete job $j \in \mathcal{J}$, regardless of the machine it runs on. In the case of uniformly related machines, we are given as input a speed f_i ($0 < f_i \leq 1$) associated with each machine i for $1 \leq i \leq m$, and it takes p_j/f_i units of time to complete job j on machine i.

We also consider two different objectives. The first is minimizing the makespan, or the final completion time of all jobs. This problem on identical machines is denoted as $P|\text{res1}, \text{prec}|C_{max}$ in the standard scheduling notation introduced in [40]. Here P denotes identical parallel machines, as opposed to uniformly related machines which is denoted by Q. Also, the resource constraint, res1, indicates that we have a single resource, and prec stands for the general precedence constraint. C_{max} indicates that the objective is to minimize the makespan. The other objective we consider is the more general goal of minimizing the total weighted completion time. In this setting there is a set of weights $\{w_j\}_{j \in \mathcal{J}}$ associated with the jobs, and the goal is to minimize $\sum_{j \in \mathcal{J}} w_j C_j$ where C_j is the completion time of job j in the final schedule. Using the same notation, we denote this problem on identical machines as $P|\text{res1}, \text{prec}| \sum_j w_j C_j$. Corresponding problems on uniformly related machines are denoted

as $Q|\text{res1}, \text{prec}|C_{max}$ and $Q|\text{res1}, \text{prec}|\sum_j w_j C_j$, respectively. Note that the minimum total weighted completion time objective ($\sum_j w_j C_j$) is more general than the minimum makespan objective (C_{max}) in the presence of precedence constraints as one could transform a C_{max} objective into a $\sum_j w_j C_j$ objective by simply setting all w_j's to be 0 and adding a "last" job j' with $p_{j'} = 0$, $s_{j'} = 0$, and $w_{j'} = 1$ which depends on every other job.

In addition, we consider these problems with release time constraints. In this case, we have a release time $r_j \in \mathbb{Z}_{\geq 0}$ associated with each job in the input such that j can only be started after time r_j. We add r_j in the middle constraint section in the scheduling notation to denote the problems with the additional release time constraint (e.g. $P|\text{res1}, \text{prec}, r_j|C_{max}$).

We also consider a generalization on identical machines with resource/run-time tradeoffs for each job. In particular, we are given, for each job $j \in \mathcal{J}$, a set of configurations $\mathcal{C}_j$ each $c \in \mathcal{C}_j$ with a processing time $\text{p}(c) \in \mathbb{Z}_{\geq 0}$ and a resource requirement $\text{s}(c) \in \mathbb{Z}_{\geq 0}$. A feasible schedule now needs to consider the resource/run-time tradeoff of different configurations, choose one $c \in \mathcal{C}_j$ for each job j, and schedule it to run for $\text{p}(c)$ amount of time consuming $\text{s}(c)$ amount of the available resource during that time. This generalization with a discrete set of resource/run-time tradeoffs for each job models a scenario encountered in real systems where the resource is *power consumption* and we present empirical results on our algorithm for this *configurable* variation.

3.2 Related Work

Precedence Constrained Scheduling. One important class of problems is scheduling subject to only precedence constraints (e.g. $P|\text{prec}|C_{max}$, $Q|\text{prec}|\sum_j w_j C_j$). Almost all variants of these problems have been studied extensively. For the case of identical parallel machines and the minimum makespan objective ($P|\text{prec}|C_{max}$), Graham's seminal list scheduling algorithm [41] gives a $(2 - 1/m)$-approximation. We will utilize this algorithm and explain some aspects of its analysis in Section 3.4.1. An almost matching $(2 - \epsilon)$-hardness of approximation result is obtained by Svensson [87] assuming a stronger version

of the Unique Game Conjecture. For the more general weighted completion time objective ($P|\text{prec}|\sum_j w_j C_j$), Hall et al. [43] gave a 7-approximation which was later improved to a 4-approximation by Munier, Queyranne and Schulz [71]. Very recently, Li [62] obtained the current best ratio of $(2 + 2\ln 2 + \epsilon)$. Of course, all the hardness results for minimizing makespan hold for minimizing the weighted total completion time; however, no stronger hardness results are known for this apparently harder problem. For related machines running at different speeds—$Q|\text{prec}|\sum_j w_j C_j$ and $Q|\text{prec}|C_{max}$—Chudak and Shmoys [30] gave an $O(\log m)$-approximation by grouping the machines into $\log m$ groups according to their speed and treating the machines in each group as identical parallel machines. Li [62] improved this ratio to $O(\log m/\log\log m)$. On the negative side, Bazzi and Norouzi-Fard [16] recently showed the problem is hard to approximate for any constant assuming the hardness of an optimization problem on k-partite graphs.

Resource Constrained Scheduling. Another overlapping set of problems is resource-constrained scheduling with one common resource (e.g., $P|\text{res1}|C_{max}$ and $Q|\text{res1}|\sum_j w_j C_j$). Garey and Graham's result in [39] implies a $(3-3/m)$-approximation for $P|\text{res1}|C_{max}$. Later, Niemeier and Wiese [73] gave the algorithm with the current best approximation ratio of $(2+\epsilon)$. Recently, Jansen et al. [48] provided an AFPTAS for the problem. Since bin packing is a special case of this problem, a simple reduction from the partition problem shows that no approximation with a ratio smaller than 3/2 is possible unless $P = NP$.

Power-aware Scheduling. In Systems literature, the resource constrained machine scheduling is usually considered in the context of power-aware scheduling when the resource is power consumption by the machines. Most power-aware algorithms assume the only configurable part of the system is processor speed. They then model the speed scaling using the standard assumption proposed by Yao, Demers, and Shenker (YDS) [93]: there is a continuous *power function* $P(s)$, such that a processor running at speed s has power consumption $P(s) = s^\alpha$ for some $\alpha > 1$. Unfortunately, this assumption ignores the fact that

real processors have both minimum and maximum speeds; i.e., in real computing systems one cannot arbitrarily increase energy efficiency ($s/P(s)$) by slowing down computation [53].

Nevertheless, many researchers have proposed extensions and improvements of the YDS algorithm. These include algorithms that account for non-zero idle-power states [3] and algorithms that consider zero power sleep states that require additional energy and time to restart computation [45]. Numerous other improvements on the original YDS algorithm exist in the scheduling literature [18, 94, 76, 9, 4, 13, 12, 25, 14].

Most approaches, however, schedule only independent jobs (i.e. no precedences) and maintain the YDS assumption about the relationship between power and speed. Pruhs, van Stee, and Uthaisombut [77] consider speed scaling to minimize total energy for multiprocessor settings with precedence constraints between jobs. Bunde assumes that jobs arrive over time and considers continuous power functions that are convex, strictly convex, or discrete [19]. He develops algorithms that minimize the makespan for uniprocessor and multiprocessor settings where the processors have a shared energy budget. This technique is not applicable to our multi-machine setting with a resource (e.g. power) bound, rather than an energy bound—i.e., a bound on the maximum *rate* of energy consumption rather than total energy consumption—as the combinations of such configurations may violate the resource (e.g. power) constraint.

Power bounds, however, are essential for future generation high-performance computing systems as noted by several recent studies [84, 17, 81]. Recent heuristic schedulers account for power by attempting to predict a workload's critical path and reduce power where it will not affect execution time [51, 66]. Patki et al. propose building HPC systems that are specifically *over-provisioned* and could easily draw much more power than is safe to deliver to the system at once [74]. Sarood et al. have shown similar results: hardware over-provisioning increases performance given a power cap [82]. These hardware over-provisioning approaches acknowledge that compute resources are no longer the primary factor limiting cluster size—power is. Furthermore, these approaches require implementing severely restrictive power

caps, where the system-level power budget is significantly below the maximum total power draw of the available machines. This scenario is precisely the realm where our proposed divide-and-conquer algorithm provides the biggest advantages over greedy approaches.

As existing scheduling implementations are heuristic in nature, Bailey et al. recently proposed a mathematical optimization framework for analyzing job schedules under a power constraint [11]. This formulation considers configurability, power constraints, and precedence constraints, but it is an offline algorithm and requires imposing a total order on the precedences representing the workload to be scheduled. For these reasons, the Bailey et al. approach is best suited for post hoc analysis of heuristic schedules, to determine how far they deviate from optimal.

Packing Problems. There is a clear connection between resource constrained scheduling and packing problems, such as bin packing and strip packing. In the case of unit processing times ($p_j = 1$), for example, $P|\text{res1}, p_j = 1|C_{max}$ is the same as the bin packing problem where we pack different sized items into bins of fixed size and try to minimize the number of bins used. In the strip packing problem, we have a fixed width strip with one open end and a finite set of rectangles. The goal is to fit all the rectangles in the strip minimizing the total height that the rectangles reach. The correspondence between strip packing and resource constrained scheduling is obvious. The width of the strip corresponds to the global resource limit and the widths and the heights of the rectangles correspond to the resource requirement and the processing time of the jobs respectively. In fact, we make use of this correspondence between the two problems in our algorithms. One difference between these two problems is that we can pack as many small width rectangles as the width of the strip allows in the strip packing problem whereas the number of jobs that can run in parallel is bounded by the number of machines in the resource constrained scheduling problem.

Malleable Job Scheduling. Malleable job scheduling with precedence constraints resembles the configurable variation of the problem we consider here in all three aspects—resource

limits, precedence constraints, and configurability. For malleable jobs, each job is allotted a number of 1 to m processors. A job's run-time is a function of the processors allotted. A feasible schedule ensures that co-scheduled jobs are allotted no more than m machines while respecting the precedence dependencies. It is often assumed that jobs' run-times are non-increasing and the work (the number of processor times the run-time) is non-decreasing with increasing processors. Under such assumptions, the problem admits constant-approximations [58, 49, 50].

While similar, malleable job scheduling differs from our problem in several key ways. Most importantly, we model the resource as a continuous variable; at any point in a feasible schedule the jobs' resource requirements may sum to any real number between 0 and the resource capacity S. In contrast, in malleable job scheduling, there are only m states a schedule can be in at any point. This discrete modeling of resource (together with the assumptions on the function of run-time) is the key for obtaining a constant-approximation for the malleable job scheduling with precedence constraints. The continuous nature of the resource in our problem renders the same methods less useful.

The malleable job problem clearly differs from that considered in this work from a systems point of view as well. Malleable job scheduling reduces configurability to simply the number of processors, which determines a job's run-time. In contrast, we do not impose any assumptions on the machine/job pairs' configuration spaces. Additionally, our configurations are not necessarily directly comparable with each other (but comparable in terms of their resource requirement). For example, one configuration may correspond to four active cores on a lower DVFS setting with no hyperthreading and two memory controllers and another one may correspond to two active cores on a higher DVFS setting with hyperthreading and a single memory controller (see Section 3.6.3).

3.3 Summary of Our Results and Techniques

3.3.1 Theoretical Results

We study identical parallel machines in Section 3.4. First, in Section 3.4.1, we consider the objective of minimizing the makespan under both resource and precedence constraints, namely the problem $P|\text{res1}, \text{prec}|C_{max}$. We prove the following.

Theorem 3.4.1. *There is a* $2 + 2\log n$*-approximation algorithm for* $P|res1, prec|C_{max}$.

To show this, we consider the two constraints separately and present a two-step algorithm for minimizing the makespan. The first step produces an intermediate approximate schedule satisfying only the precedence constraints with a loss of factor of 2 in the approximation. The second step uses a divide and conquer algorithm (inspired by [8]) to stretch the intermediate schedule so that it also satisfies the resource constraint. We generalize this result for the version of the problem with release times.

Theorem 3.4.8. *There is a* $2 + 4\log n$*-approximation algorithm for* $P|res1, prec, r_j|C_{max}$.

In Section 3.4.2, we use the algorithm from Section 3.4.1 as a subroutine to obtain a similar result under minimum weighted completion time objective:

Theorem 3.4.9. *There is an* $O(\log n)$*-approximation algorithm for* $P|res1, prec, r_j|\sum_j w_j C_j$

To obtain this result, we extend the general framework of [43] and [30] with a *novel* LP relaxation for $P|\text{res1}, \text{prec}, r_j|\sum_j w_j C_j$. In this framework, we divide the time horizon into geometrically increasing intervals. Using our new linear program, we obtain the "approximate completion interval" for each job. Then we show that if we consider, for each interval, the set of jobs completing in the same interval as a separate instance of $P|\text{res1}, \text{prec}, r_j|C_{max}$ problem and schedule them in a separate fragment using the makespan minimizing algorithm we obtain in Section 3.4.1, the concatenation of these fragments also gives a good approximation to the minimum total weighted completion time objective ($\sum_j w_j C_j$) for the original

instance. Our core technique is our time-interval-indexed LP where we carefully incorporate the resource requirements into the LP to guarantee a reasonable total resource requirement by the set of jobs completing in any given interval. This allows us to bound the makespan given by our algorithm from Section 3.4.1 for each $P|\text{res1}, \text{prec}, r_j|C_{max}$ instance.

In Section 3.5, we build on our previous techniques to show that they can be modified to work together with the methods in [30] to get an $O(\log m \log n)$-approximation for scheduling on uniformly related machines subject to simultaneous resource, precedence, and release times constraints with weighted completion time objective. In particular, we state a simple generalization of the time-interval-indexed LP used in the previous section (and also used in [30] without the resource LP constraints) to machines with different speeds. We use the LP solution to get the approximate completion time intervals as we did for identical machines setting as well as job to machine assignments. Then, we argue that the first step of our two-step makespan minimizing algorithm can be replaced by the $O(\log m)$-approximation algorithm for $Q|\text{prec}, r_j|C_{max}$ given in [30]. Moreover, the second step of our algorithm will respect these job to machine assignments. If we begin with an optimal solution to the LP, we show that these integral job to machine assignments are close enough to the fractional assignments given by the LP solution.

Theorem 3.5.1. *There is an $O(\log m \log n)$-approximation algorithm for $Q|$res1, prec, $r_j|$ $\sum_j w_j C_j$.*

These are the first algorithms with non-trivial approximation ratios for this class of scheduling problems where a resource constraint and general precedence constraints need to be satisfied together.

3.3.2 Empirical Results

We propose our algorithm as a solution to an emerging problem in High Performance Computing: precedence constrained scheduling under a *power-cap*. Here our resource is

power. We evaluate our algorithm in a simulation environment against three state-of-the-art algorithms using greedy strategies. We use precedence constraints and power/run-time data collected from real HPC applications. We show that our divide-and-conquer approach outperforms greedy approaches especially well in severely power-restricted systems with significant energy savings and improving performance up to 75%.

We use simulation infrastructure for this evaluation because our goal is to compare fundamentally different algorithmic strategies (greedy versus divide-and-conquer), rather than to evaluate practical implementation issues. All modern job management systems (of which we are aware) are based on greedy strategies. Performance-aware schedulers minimize run-time respecting precedence constraints—on both distributed (e.g. Spark [95] and MapReduce [32]) and multi-core systems (e.g. Cilk++ [57] and TBB [79])—based on greedy strategies derived from that first proposed and analyzed by Graham [41]. State-of-the-art power-aware schedulers also use greedy schedules to maximize throughput for independent jobs [86, 83, 75, 44]. Whether constrained by precedence or power, all these recent scheduling efforts share a common algorithmic foundation: *they adopt greedy strategies and innovate for practical concerns*. In this thesis, we present a fundamentally different algorithmic strategy: divide-and-conquer. Our simulation-based evaluation factors out implementation concerns to focus on the potential benefits of divide-and-conquer over the greedy strategy that underlies all known schedulers.

3.4 Identical Parallel Machines

3.4.1 The Minimum Makespan Objective

In this section, we present an $O(\log n)$-approximation algorithm for the problem of minimizing the makespan under both resource and precedence constraints without resource/run-time tradeoffs — i.e. jobs have fixed run-times and resources.

Theorem 3.4.1. *There is a* $2 + 2\log n$*-approximation algorithm for* $P|res1, prec|C_{max}$.

Given an instance $(\mathcal{J}, m, \{p_j\}_{j\in\mathcal{J}}, S, \{s_j\}_{j\in\mathcal{J}}, \prec)$ of $P|\text{res1}, \text{prec}|C_{max}$, we let OPT denote the makespan of a minimum makespan schedule of this instance. Our algorithm solves the problem in two steps by handling its precedence and resource constraints separately. First, we consider the corresponding $P|\text{prec}|C_{max}$ instance where we drop the resource requirement (i.e. $(\mathcal{J}, m, \{p_j\}_{j\in\mathcal{J}}, \prec)$). In his seminal work, Graham [41] presents an online machine-driven list scheduling algorithm for this problem. His algorithm greedily considers the jobs in some arbitrary extension of the partial order $\prec$ to a total order (i.e. a list). As soon as a machine is idle, the algorithm schedules the next available job (i.e. all its predecessors are finished) in the list on that machine. In the analysis, he uses two standard lower bounds for the value of OPT:

Lemma 3.4.2 (Load Bound)**.** $\frac{1}{m}\sum_{j\in\mathcal{J}} p_j \leq OPT$.

Lemma 3.4.3 (Chain Bound)**.** $\max_{\mathcal{C} \text{ is a chain}} \sum_{j\in\mathcal{C}} p_j \leq OPT$.

The Load Bound is implied by the observation that a perfectly balanced schedule with no idle time would have a makespan of $\frac{1}{m}\sum_{j\in\mathcal{J}} p_j$. Also note that the total processing time of any "chain" of precedence constraints such as $j_1 \prec j_2 \prec \cdots \prec j_k$ is a lower bound on the makespan of any schedule. His analysis charges the time intervals where all the machines are busy on the Load Bound and the time intervals where some machines are idle on the Chain Bound.

Note that the total resource required to complete a job $j \in \mathcal{J}$ is its resource requirement integrated over the time it takes to complete the job: $\int_0^{p_j} s_j \,\mathrm{d}t = s_j p_j$. We denote this value by $\mathsf{RB}(j)$. We define $\mathsf{RB}(J)$ for any subset $J \subseteq \mathcal{J}$ of jobs as the sum of the total resource required for jobs in J (i.e. $\mathsf{RB}(J) = \sum_{j\in J} \mathsf{RB}(j)$). Our algorithm uses a third lower bound derived by comparing the resource available in a makespan and total resource required by all jobs.

Lemma 3.4.4 (Resource Bound)**.** $\mathsf{RB}(\mathcal{J})/S \leq OPT$.

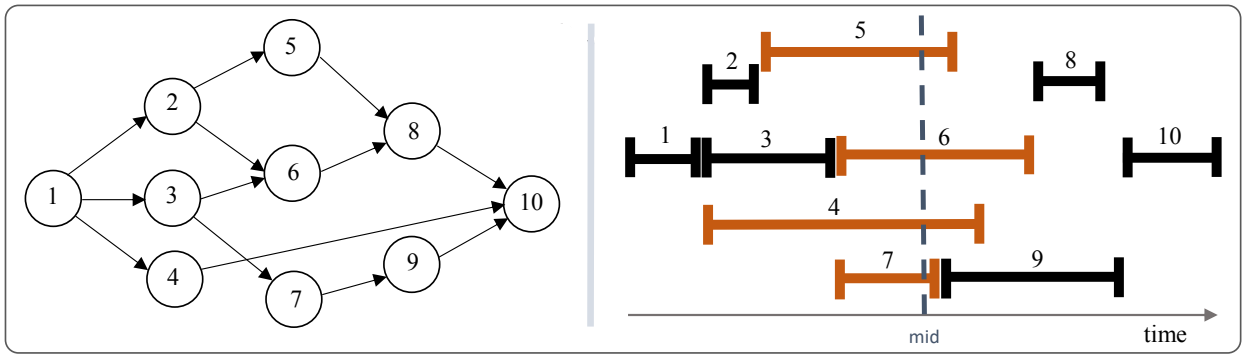


Figure 3.1: The intermediate schedule (right) obtained in the first step from the DAG (left) representing the precedence constraints prec. The intermediate schedule satisfies the precedences, but not necessarily the power constraints.

Our algorithm consists of two steps. First, we run Graham's list scheduling algorithm on our instance after we drop the resource constraints and get an intermediate schedule satisfying the precedence constraints. Let $\mathsf{LS}_{\mathcal{J}}$ denote the makespan of this schedule and $\mathsf{LS}(j)$ denote the completion time of a job $j \in \mathcal{J}$ in it. Furthermore, for any subset of jobs $J \subseteq \mathcal{J}$, we let LS_J denote the length of the shortest interval in the intermediate schedule that fully contains all the jobs in J. Graham [41] shows this makespan is bounded by the sum of the Load and the Chain Bounds $\mathsf{LS}_{\mathcal{J}} \leq \frac{1}{m}\sum_{j\in\mathcal{J}} p_j + \max_{\mathcal{C} \text{ is a chain}} \sum_{j\in\mathcal{C}} p_j \leq 2 \cdot OPT$.[1]

The schedule we get by running Graham's list scheduling on the corresponding $P|\text{prec}|C_{max}$ instance may violate the resource constraints of the original instance. In the second step of the algorithm, we run a divide and conquer algorithm on this schedule to make it satisfy the resource constraints as well without disturbing the precedences already satisfied after our first step. We lose a factor of 2 in the approximation due to Graham's algorithm and an $O(\log n)$-factor in the second step. In the rest of this section, we explain and analyze the second step of the algorithm in detail.

In order to incorporate the resource constraint into the schedule obtained in the first step, we run a divide and conquer algorithm similar to the one employed by Augustine et al. [8] for strip packing. This algorithm (Algorithm 1: **DS**) will need to use a subroutine for scheduling

1. In fact, [41] shows a slightly stronger bound of $(2 - 1/m)\,OPT$.

at most m jobs $J \subseteq \mathcal{J}$ with resource constraints that have no precedence constraints among them on m machines. We denote this subroutine by Next-Fit Decreasing-Height: $\mathbf{NFDH}(J)$ and it guarantees a makespan that is bounded by $\mathbf{NFDH}(J) \leq 2\mathsf{RB}(J)/S + \max_{j \in J} p_j$. A simple greedy algorithm that schedules the jobs respecting their resource constraints in the order of non-increasing processing time will have this guarantee. Next-Fit Decreasing-Height algorithm for strip packing analyzed in [38] is one such algorithm when applied to scheduling m jobs on m machines with resource constraints and no precedence constraints.

Algorithm 1 $\mathbf{DS}(J, B, E)$ Divide and Schedule

Input: A subset of jobs $J \subseteq \mathcal{J}$ and the beginning B and the end E times of J in the schedule of the first step

Output: A makespan y for a feasible schedule of J

1: **if** $J = \emptyset$ **then**
2: **return** $\emptyset$
3: **end if**
4: Let mid be a time-point s.t. $|J_{bef}| \leq |J|/2$ and $|J_{aft}| \leq |J|/2$ w.r.t. definitions below
5: $J_{bef} \leftarrow \{j \in J \mid \mathsf{LS}(j) < mid\}$
6: $J_{mid} \leftarrow \{j \in J \mid \mathsf{LS}(j) - p_j < mid \text{ and } \mathsf{LS}(j) \geq mid\}$
7: $J_{aft} \leftarrow \{j \in J \mid \mathsf{LS}(j) - p_j \geq mid\}$
8: $S_{bef} \leftarrow \mathbf{DS}(J_{bef})$
9: $S_{mid} \leftarrow \mathbf{NFDH}(J_{mid})$
10: $S_{aft} \leftarrow \mathbf{DS}(J_{aft})$
11: **return** Concatenation of schedules S_{bef}, S_{mid}, and S_{aft}

The point mid is defined as a point which leaves at most half of the jobs on each side. We show there exists a point satisfying this definition.

Lemma 3.4.5. *There is at least one time point mid such that, when **DS** partitions the jobs in intermediate schedule around mid, $|J_{bef}| \leq |J|/2$ and $|J_{aft}| \leq |J|/2$.*

Proof. Consider a sweep starting from the end point of the intermediate schedule towards its starting point and the first critical point cp where $|J_{bef}| \leq |J|/2$. Immediately to the right of cp, let the size J_{bef} be $\lfloor |J|/2 \rfloor + i$ for some $i \geq 1$. On cp, $|J_{mid}| \geq i$ and $|J_{bef}| = \lfloor |J|/2 \rfloor + i - |J_{mid}|$. Then, $|J_{aft}| = |J| - |J_{mid}| - |J_{bef}| = |J| - |J_{mid}| - \lfloor |J|/2 \rfloor - i + |J_{mid}| = \lceil |J|/2 \rceil - i \leq |J|/2$. Since there are $O(|J|)$ critical points, such a sweep can be done efficiently. □

The algorithm **DS** is initially called with all jobs $\mathcal{J}$. It takes the set of jobs J_{mid} scheduled to cross the point *mid* in the intermediate schedule obtained in the first step. It schedules these jobs in a separate time fragment and concatenates it with schedules obtained recursively on the jobs before, J_{bef}, and the jobs after, J_{aft}. We need the following lemma to justify the initial condition required to call $\mathbf{NFDH}(J_{mid})$.

Lemma 3.4.6. *The jobs in J_{mid} have no precedence constraints among them and $1 \leq |J_{mid}| \leq m$.*

Proof. Since the jobs in $J_{mid} = \{j \in J \mid \mathsf{LS}(j) - p_j \leq mid \text{ and } \mathsf{LS}(j) > mid\}$ are scheduled to be running all in parallel at some time point (*mid*) by the intermediate schedule obtained in the first step, none of them can depend on the other as this would make the initial schedule from the first step infeasible. The intermediate schedule has no time point where all the machines are idle due to the nature of Graham's list scheduling as it schedules the next job on an idle machine given none of its predecessors is still running. This implies $|J_{mid}| \geq 1$. This schedule may violate the resource constraints but it will be a feasible schedule for the corresponding $P|\text{prec}|C_{max}$ problem. Thus, we will have only at most m jobs running at any time in this schedule. □

$\mathbf{NFDH}(J_{mid})$ charges bulk of the length of the schedule it returns to the Resource Bound except for a loss bounded by the length of the longest job in J_{mid}, $\max_{j \in J_{mid}} p_j$. Thus, in any given level of the recursion tree, the total loss across all recursive calls in this level is at most an additive factor of $O(OPT)$ resulting from **NFDH** subroutine. Since the algorithm terminates in at most $\lceil \log n \rceil$ levels, we get an $O(\log n)$ approximation. Lemma 3.4.7 proves this formally and gives a bound on our initial call to Divide and Schedule algorithm: $\mathbf{DS}(\mathcal{J}) \leq 2\mathsf{RB}(\mathcal{J})/S + \mathsf{LS}_{\mathcal{J}} \log(|\mathcal{J}|) \leq (2 + 2\log n)OPT$. This completes the proof of Theorem 3.4.1.

Lemma 3.4.7. $\boldsymbol{DS}(J) \leq 2\mathsf{RB}(J)/S + \mathsf{LS}_J \log |J|$

Proof. We prove this by induction on the size of J. When $|J| = 0$, the case is handled by the if statement and the inequality holds. The following hold by induction ($|J_{bef}|, |J_{aft}| < |J|$ by Lemma 3.4.6).

$$|S_{bef}| = \mathbf{DS}(J_{bef}) \leq \frac{2}{S}\mathsf{RB}(J_{bef}) + \mathsf{LS}_{J_{bef}} \log |J_{bef}|$$
$$|S_{aft}| = \mathbf{DS}(J_{aft}) \leq \frac{2}{S}\mathsf{RB}(J_{aft}) + \mathsf{LS}_{J_{aft}} \log |J_{aft}|$$

We also have the bound on the length of the schedule obtained by **NFDH** on J_{mid}

$$|S_{mid}| = \mathbf{NFDH}(J_{mid}) = \frac{2}{S}\mathsf{RB}(J_{mid}) + \max_{j \in J_{mid}} p_j.$$

Then,

$$\begin{aligned}
\mathbf{DS}(J) &= |S_{beg}| + |S_{mid}| + |S_{aft}| \\
&\leq \frac{2}{S}\mathsf{RB}(J_{bef}) + \mathsf{LS}_{J_{bef}} \log |J_{bef}| + \frac{2}{S}\mathsf{RB}(J_{mid}) + \max_{j \in J_{mid}} p_j \\
&\quad + \frac{2}{S}\mathsf{RB}(J_{aft}) + \mathsf{LS}_{J_{aft}} \log |J_{aft}| \\
&\leq \frac{2}{S}\left(\mathsf{RB}(J_{bef}) + \mathsf{RB}(J_{mid}) + \mathsf{RB}(J_{aft})\right) + \mathsf{LS}_{J_{bef}} \log \frac{|J|}{2} + \mathsf{LS}_{J_{aft}} \log \frac{|J|}{2} + \max_{j \in J_{mid}} p_j \\
&\leq \frac{2}{S}\mathsf{RB}(J) + \left(\mathsf{LS}_{J_{bef}} + \mathsf{LS}_{J_{aft}}\right) \log \frac{|J|}{2} + \mathsf{LS}_J \\
&\leq \frac{2}{S}\mathsf{RB}(J) + \mathsf{LS}_J \left(\log \frac{|J|}{2} + 1\right) \\
&= \frac{2}{S}\mathsf{RB}(J) + \mathsf{LS}_J \log |J|
\end{aligned}$$

The second inequality holds because $|J_{bef}|, |J_{aft}| \leq |J|/2$ by the definition of *mid* in Line 4 in the algorithm **DS**. Third inequality holds because $\max_{j \in J_{mid}} p_j$ is a part of the intermediate schedule and it is at most LS_J. J_{bef} and J_{aft} are disjoint and the intervals containing them in the intermediate schedule are disjoint, so the last inequality is justified by the fact that $\mathsf{LS}_{J_{bef}} + \mathsf{LS}_{J_{aft}} \leq \mathsf{LS}_J$. □

The first step of our algorithm can be extended to accommodate release times constraints:

Theorem 3.4.8. *There is a* $2 + 4\log n$*-approximation algorithm for* $P|res1, prec, r_j|C_{max}$.

In addition to resource and precedence constraints, each job $j \in \mathcal{J}$ now has a release time r_j such that j can only be started after time r_j ($P|\text{res1}, \text{prec}, r_j|C_{max}$). Munier, Queyranne and Schulz [71, 78] provide a 4-approximation for $P|\text{prec}, r_j|C_{max}$. We replace Graham's list scheduling in the first step of our algorithm with their 4-approximation algorithm to get a similar bound for the problem with all three constraints. Note that, after the first step, we obtain a schedule that satisfies the release times and precedence constraints. The second step of our algorithm can easily be made to never schedule a job before its starting time in the intermediate schedule obtained the first step by forcing J_{mid}'s fragment to start at $\max\{mid, |S_{bef}|\}$.

We note that an $o(\log n)$-approximation is not possible using only the Load, Chain, and Resource Bounds. The bottleneck is in the second step where we use divide-and-conquer. The gap example in [8] for strip packing shows one needs stronger lower bounds on OPT than "the area bound" (Resource Bound in our setting) and "the longest chain of rectangles bound" (Chain Bound in our setting) for an $o(\log n)$ approximation. Their example can easily be translated into a scheduling instance with $m = \log n$ machines. Setting $m = \log n$ allows any solution to their example be interpreted as a scheduling solution because the example has at most $\log n$ parallel precedence chains at any point. Also, the Load Bound of the translated instance is at most $\Theta(1)$, where $OPT = \Omega(\log n)$.[2] Thus, for both the makespan and the weighted completion time objectives, one needs stronger lower bounds on OPT than any combination of the three bounds we use for a better approximation ratio.

2. See [8] and their illustrations for a detailed description of the gap example.

3.4.2 The Minimum Weighted Completion Time Objective

In this section, we generalize our result by giving an $O(\log n)$-approximation algorithm for the minimum total weighted completion time objective. In addition to processing time p_j, resource requirement s_j, and release time r_j, we now have a weight w_j associated with each job $j \in \mathcal{J}$ and our goal is to minimize the total weighted completion time $\sum_j w_j C_j$, where C_j is the completion time of j in the final schedule. This problem is denoted as $P|\text{res1}, \text{prec}, r_j|\sum_j w_j C_j$.

Theorem 3.4.9. *There is an $O(\log n)$-approximation algorithm for $P|res1, prec, r_j|\sum_j w_j C_j$*

We first reduce the problem of minimizing weighted completion time ($\sum_j w_j C_j$) to a set of smaller problems with the objective of minimizing the makespan (C_{max}). We then use our $O(\log n)$-approximation algorithm from Section 3.4.1 for the minimum makespan objective as a subroutine on these problems. In the reduction, we use the general framework of Hall et al. [43] (see also, for example, [30] for an application of this framework in uniformly related machines setup).

We start with a trivial upper bound 2^L on the length of an optimal schedule, where $L = \lceil \log(n \cdot \max_{j \in \mathcal{J}}(r_j + p_j)) \rceil$, and divide the time horizon into a set of geometrically increasing intervals $[1, 2], (2, 4], (4, 8], \cdots, (2^{L-1}, 2^L]$. For the problem with only precedence and release time constraints ($P|\text{prec}, r_j|\sum_j w_j C_j$), Hall et al. [43] argue that for each job j, if we are given the interval in which it completes in the optimal schedule, we can get an approximate schedule based on this information. All the jobs completing in $(2^{l-1}, 2^l]$ in the optimal schedule can be scheduled to start and complete in $(2^{l+1}, 2^{l+2}]$ using a makespan minimizing algorithm on this subset of jobs as a subroutine. This results in an 8-approximation for the weighted completion time objective. Since the completion intervals in an optimal schedule are not known, Hall et al. [43] use an LP relaxation to obtain approximate values for the completion intervals. The makespan of the makespan minimizing algorithm they use for $P|\text{prec}, r_j|C_{max}$ depends only on the Load Bound and the Chain Bound, which are both easy

to bound in the same LP where they get the completion intervals. However, in our setting, we will need stronger guarantees when obtaining the completion intervals with an LP than the previously used LPs in the same framework can provide. In particular, we will need that the jobs completing in a given interval have total resource requirement that is comparable to the resource available up to that interval. In this way, we introduce the following linear programming relaxation for the $P|\text{res1}, \text{prec}, r_j|\sum_j w_j C_j$ problem which makes sure that we get a good estimate on the completion interval of each job and, at the same time, guarantees that the total resource requirement by jobs completing in some interval is not "too much". We let $[a]$ denote the set $\{1, 2, \cdots, a\}$ for a positive integer a.

$$\min \quad \sum_{j \in \mathcal{J}} w_j C_j \qquad \text{s.t.} \tag{LP_P}$$

$$\sum_{i=1}^{m} \sum_{t=1}^{L} x_{ijt} = 1, \qquad \forall j \in \mathcal{J}; \tag{3.1}$$

$$p_j \leq C_j - r_j, \qquad \forall j \in \mathcal{J}; \tag{3.2}$$

$$p_j \leq C_j - C_{j'}, \qquad \forall j' \prec j; \tag{3.3}$$

$$\sum_{t=1}^{L} 2^{t-1} \sum_{i=1}^{m} x_{ijt} \leq C_j, \qquad \forall j \in \mathcal{J}; \tag{3.4}$$

$$\sum_{j \in \mathcal{J}} p_j \sum_{t=1}^{l} x_{ijt} \leq 2^l, \qquad \forall i \in [m], l \in [L]; \tag{3.5}$$

$$\sum_{i=1}^{m} \sum_{t=1}^{l} x_{ijt} \leq \sum_{i=1}^{m} \sum_{t=1}^{l} x_{ij't}, \qquad \forall j' \prec j, l \in [L]; \tag{3.6}$$

$$\sum_{j \in \mathcal{J}} p_j s_j \sum_{i=1}^{m} \sum_{t=1}^{l} x_{ijt} \leq 2^l S, \qquad \forall l \in [L]; \tag{3.7}$$

$$x_{ijt} \geq 0, \qquad \forall i \in [m], j \in \mathcal{J}, t \in [L]; \tag{3.8}$$

In LP_P, we have two sets of variables: a set of real valued variables for the completion times of the jobs $\{C_j\}_{j \in \mathcal{J}}$ and a set of decision variables $\{x_{ijt}\}_{i \in [m], j \in \mathcal{J}, t \in [L]}$ that take 0-1 values

in an integral solution with $x_{ijt} = 1$ implying that the job j completed on machine i in the time interval $(2^{t-1}, 2^t]$. Constraint 3.1 says a job needs to complete on some machine and in some interval. Constraints 3.2 and 3.3 make sure that the completion times are not infeasible with respect to release times and the precedence constraints. Constraint 3.4 says that the completion time of a job needs to be later than the time point marking the start of the time interval in which the job completes. Constraint 3.5 ensures that the total processing time of the jobs completing on machine i in the first l intervals is at most the time point marking the end of the lth interval. Constraint 3.6 ensures that a job j depending on another job j' must complete in j''s completion interval or later. Finally, Constraint 3.7 is how we guarantee that the total resource requirement of the jobs completing in the first l intervals is at most the total amount of resource available in these intervals.

Let $\{\tilde{x}_{ijt}\}$ and $\{\tilde{C}_j\}$ be a solution to LP_P. We now describe how to obtain the approximate completion intervals from this fractional LP solution and the smaller problems with minimum makespan objective by partitioning the set of jobs with respect to these completion intervals. Let $\ell_1(j)$ be the first interval l where sum of job j's variables x_{ijt} across all machines exceed 1/2 (i.e. minimum l s.t. $\sum_{t=1}^{l}\sum_{i=1}^{m}\tilde{x}_{ijt} \geq 1/2$). Also, define $\ell_2(j)$ to be the interval containing job j's completion time (i.e. minimum l s.t. $C_j \leq 2^l$). We let $\ell(j) = \max\{\ell_1(j), \ell_2(j)\}$. Define, for each $l \in [L]$, $J_l = \{j \in \mathcal{J} : \ell(j) = l\} \subseteq \mathcal{J}$ to be the subset of jobs that "complete" in the lth interval (jobs with $\ell(j) = l$). Note that the sets $J_1, J_2, \cdots, J_L$ are disjoint and they partition the set of jobs $\mathcal{J}$.

Next, we consider each J_l as a separate instance of $P|\text{res1}, \text{prec}|C_{max}$ problem and run our makespan minimizing algorithm from Section 3.4.1 on the smaller instance $(J_l, m, \{p_j\}_j, S, \{s_j\}_j, \prec_{J_l})$ where $\prec_{J_l} \subseteq \prec$ is the subset of the precedence constraints that are between the jobs in J_l.

Lemma 3.4.10. *Algorithm in Section 3.4.1 on the instance* $(J_l, m, \{p_j\}_{j\in J_l}, S, \{s_j\}_{j\in J_l}, \prec_{J_l})$ *returns a feasible schedule of length* $(4 + 3\log|J_l|)\,2^l$.

Proof. By summing up Constraint 3.5 over all machines, we get $\sum_{j\in\mathcal{J}} p_j \sum_{i=1}^{m}\sum_{t=1}^{l}\tilde{x}_{ijt} \leq$

$m \cdot 2^l$. Since $\sum_{i=1}^{m} \sum_{t=1}^{\ell_1(j)} \tilde{x}_{ijt} \geq 1/2$ and $l = \ell(j) \geq \ell_1(j)$, the Load Bound ($\frac{1}{m} \sum_{j \in J_l} p_j$) of the jobs in J_l is bounded by 2^{l+1}. Similarly, since we have Constraint 3.3 and that $\max_{j \in J_l} \tilde{C}_j \leq \max_{j \in J_l} 2^{\ell_2(j)} \leq 2^l$, their Chain Bound is bounded by 2^l. Thus, Graham's algorithm in the first step will return a schedule of length at most the sum of Load and Chain Bounds, which is at most $3 \cdot 2^l$. This bounds $\mathsf{LS}_{J_l} \leq 3 \cdot 2^l$.

We now bound the Resource Bound of J_l. Consider a job j with $\ell(j) = l$ (i.e. $j \in J_l$). If $\ell_1(j) = l$, then $\sum_{t=1}^{l} \sum_{i=1}^{m} \tilde{x}_{ijt} \geq 1/2$ by definition of $\ell_1(j)$. Otherwise, $\ell_2(j) = l > \ell_1(j)$ and $\sum_{t=1}^{l} \sum_{i=1}^{m} \tilde{x}_{ijt} \geq \sum_{t=1}^{\ell_1(j)} \sum_{i=1}^{m} \tilde{x}_{ijt} \geq 1/2$. Then the resource requirement of J_l satisfies:

$$\mathsf{RB}(J_l) = \sum_{j \in J_l} p_j s_j \leq 2 \sum_{j \in J_l} p_j s_j \sum_{t=1}^{l} \sum_{i=1}^{m} \tilde{x}_{ijt} \leq 2 \sum_{j \in \mathcal{J}} p_j s_j \sum_{t=1}^{l} \sum_{i=1}^{m} \tilde{x}_{ijt} \leq 2^{l+1} S$$

The first inequality follows from $\sum_{t=1}^{l} \sum_{i=1}^{m} \tilde{x}_{ijt} \geq 1/2$ for all $j \in J_l$ and the last from Constraint 3.7. Thus, $\mathsf{RB}(J_l)/S \leq 2^{l+1}$ and we get a schedule of length at most $(4 + 3 \log |J_l|)\, 2^l$.

□

Using Lemma 3.4.10, we schedule, for each $l \in [L]$, the jobs in J_l to start after $(4 + 3 \log |J_l|)(1 + 2 + 4 \cdots + 2^{l-1})$ and complete before $(4 + 3 \log |J_l|)\,(1 + 2 + 4 \cdots + 2^l)$.

Lemma 3.4.11. *The resulting schedule satisfies resource, precedence, and release time constraints.*

Proof. Since partitions $J_1, J_2, \cdots, J_L$ are scheduled in separate disjoint fragments, jobs running on different machines at the same time are always from the same partition. The schedule for each partition is obtained by running the algorithm of Section 3.4.1. Thus, resource constraints are satisfied.

Similarly, the precedence constraints among jobs in J_l for any l will be satisfied. We need to show that, for jobs $j', j \in \mathcal{J}$ and $j' \prec j$, we have $\ell(j') \leq \ell(j)$. First consider the case $\ell(j') = \ell_1(j')$. By Constraint 3.6, this implies $\ell(j) \geq \ell_1(j) \geq \ell_1(j') = \ell(j')$. Now, consider

$\ell(j') = \ell_2(j')$. Constraint 3.3 makes sure that $\ell(j) \geq \ell_2(j) \geq \ell_2(j') = \ell(j')$.

If $j \in J_l$, $2^l = 2^{\ell(j)} \geq 2^{\ell_2(j)} \geq \tilde{C}_j$. In our schedule, j is started after $(4 + 3\log|J_l|)(1 + 2 + 4 \cdots + 2^{l-1}) > 2^{l+1} \geq \tilde{C}_j \geq r_j$. The last inequality is due to Constraint 3.2. □

Next lemma completes the proof of Theorem 3.4.9.

Lemma 3.4.12. *The resulting schedule has weighted completion time within* $32 + 24\log n$ *factor of the* LP_P *value* $\sum_{j\in\mathcal{J}} w_j\tilde{C}_j$.

Proof. Consider a job j with $\ell(j) = l$ (i.e. $j \in J_l$). Jobs in J_l all complete before $(4 + 3\log n)(1 + 2 + 4 \cdots + 2^l) \leq (32 + 24\log n)2^{l-2}$. Now, we show that $2^{l-2} \leq \tilde{C}_j$.

If $\ell(j) = \ell_2(j) = l$, then $2^{l-1} \leq \tilde{C}_j$ by definition of ℓ_2. Otherwise, $\ell(j) = \ell_1(j) = l$. Then,

$$2^{l-2} \leq 2^{l-1}\sum_{t=l}^{L}\sum_{i=1}^{m}\tilde{x}_{ijt} \leq \sum_{t=l}^{L}2^{t-1}\sum_{i=1}^{m}\tilde{x}_{ijt} \leq \sum_{t=1}^{L}2^{t-1}\sum_{i=1}^{m}\tilde{x}_{ijt} \leq \tilde{C}_j$$

The first inequality follows from the definition of $\ell_1(j) = l$ and Constraint 3.1. The last inequality is due to Constraint 3.4. □

3.5 Uniformly Related Machines

In this section, we describe our $O(\log n \log m)$-approximation algorithm for the problem of scheduling jobs on uniformly related machines (i.e. machines running at different speeds) under resource, precedence, and release time constraints. Again, the objective is to minimize the more general total weighted completion time. This problem is denoted as $Q|\text{res1}, \text{prec}, r_j|\sum_j w_jC_j$.

Theorem 3.5.1. *There is an* $O(\log m \log n)$*-approximation algorithm for* $Q|res1,\ prec,\ r_j|\sum_j w_jC_j$.

Now, we have a speed f_i associated with each machine $i \in [m]$ in the input and it takes p_j/f_i amount of time to complete job $j \in \mathcal{J}$ on machine $i \in [m]$. We note that the Load,

Chain, and Resource Bounds do not only depend on the set of jobs anymore, but also on the job to machine assignments in a solution (in particular, an optimal solution). This is because the processing time p_j/f_i of a job j now depends on the machine i it is assigned to. Note that this processing time does not change between machines running at the same speed. Thus we will be more interested in the speed that a job is assigned to than the particular machine. We let K be the number of distinct speeds and $\mathrm{f}_1, \mathrm{f}_2, \cdots, \mathrm{f}_K$ be all the distinct speeds among all m machines. We also let m_k be the number of machines with speed f_k for each $k \in [K]$. We start by solving a time-interval indexed linear program similar to the one we used in Section 3.4.2 but incorporates the different machine speeds in the LP.

$$\min \quad \sum_{j \in \mathcal{J}} w_j C_j \qquad \text{s.t.} \tag{LP_Q}$$

$$\sum_{k=1}^{K} \sum_{t=1}^{L} x_{kjt} = 1, \qquad \forall j \in \mathcal{J}; \tag{3.9}$$

$$\sum_{k=1}^{K} \frac{p_j}{\mathrm{f}_k} \sum_{t=1}^{L} x_{kjt} \leq C_j - r_j, \qquad \forall j \in \mathcal{J}; \tag{3.10}$$

$$\sum_{k=1}^{K} \frac{p_j}{\mathrm{f}_k} \sum_{t=1}^{L} x_{kjt} \leq C_j - C_{j'}, \qquad \forall j' \prec j; \tag{3.11}$$

$$\sum_{t=1}^{L} 2^{t-1} \sum_{k=1}^{K} x_{kjt} \leq C_j, \qquad \forall j \in \mathcal{J}; \tag{3.12}$$

$$\frac{1}{\mathrm{f}_k m_k} \sum_{j \in \mathcal{J}} p_j \sum_{t=1}^{l} x_{kjt} \leq 2^l, \qquad \forall k \in [K], l \in [L]; \tag{3.13}$$

$$\sum_{k=1}^{K} \sum_{t=1}^{l} x_{kjt} \leq \sum_{k=1}^{K} \sum_{t=1}^{l} x_{kj't}, \qquad \forall j' \prec j, l \in [L]; \tag{3.14}$$

$$\sum_{j \in \mathcal{J}} \sum_{k=1}^{K} \frac{p_j}{\mathrm{f}_k} s_j \sum_{t=1}^{l} x_{kjt} \leq 2^l S, \qquad \forall l \in [L]; \tag{3.15}$$

$$x_{kjt} \geq 0, \qquad \forall k \in [K], j \in \mathcal{J}, t \in [L]; \tag{3.16}$$

Constraints and variables of LP_Q are similar to the ones in LP_P of Section 3.4.2. We still have the real-valued completion time variables $\{C_j\}_{j\in\mathcal{J}}$. However, we are now interested in the distinct speed group a job is assigned to rather than the particular machine. In this way, we modify the decision variables as $\{x_{kjt}\}_{k\in[K],j\in\mathcal{J},t\in[L]}$ where the first index k now indicates the speed group among the machines. In an integral solution, $x_{kjt} = 1$ would stand for the job j completing in the time interval $(2^{t-1}, 2^t]$ on some machine with speed f_k. Since the processing time p_j of a job j now depends on the speed it runs on, we replace p_j in the constraints by $\sum_{k=1}^{K} p_j/\mathrm{f}_k \sum_{t=1}^{L} x_{kjt}$ which essentially is the average processing time of j with respect to fractional speed assignments of j. We make the required change to Constraints 3.2, 3.3, 3.5, and 3.7 to obtain corresponding Constraints 3.10, 3.11, 3.13, and 3.15.

Similar to what we did in the identical machine setting, our algorithm will partition the set of jobs according to approximate completion intervals obtained from LP_Q. Then it will obtain a set of smaller $Q|\text{res1}, \text{prec}, r_j|C_{max}$ instances from the original $Q|\text{res1}, \text{prec}, r_j|\sum_j w_j C_j$ instance. We solve each minimum makespan instance again in two steps by considering the $Q|\text{prec}, r_j|C_{max}$ instance in the first step and handling the resources in the second step. The bound on the makespan minimizing algorithm of Section 3.4.1 is analyzed in terms of Load, Chain, and Resource Bounds. However, as we have indicated above, we do not have well-defined Load, Chain, and Resource Bounds in the case of machines running at different speeds. In order to use and analyze a similar two-step algorithm, we use the solution to LP_Q to first fix job to machine speed assignments, then show that the Load, Chain, and Resource Bounds under these assignments are comparable to the LP_Q value. We adapt the speed-based list scheduling algorithm of [30] as our first step for solving $Q|\text{prec}, r_j|C_{max}$ and finally we apply the Divide and Schedule procedure of Section 3.4.1 as our second step.

Let $\{\tilde{x}_{kjt}\}$ and $\{\tilde{C}_j\}$ be a solution to LP_Q. We obtain "completion intervals" ($\ell(j)$'s) for all the jobs and partition $\mathcal{J}$ by defining J_l for $1 \le l \le L$ in the same way as we did in

Section 3.4.2. We let $\ell(j) = \max\{\ell_1(j), \ell_2(j)\}$ where $\ell_1(j)$ is defined as the minimum l s.t. $\sum_{t=1}^{l}\sum_{k=1}^{K}\tilde{x}_{kjt} \geq 1/2$ and $\ell_2(j)$ as the minimum l s.t. $\tilde{C}_j \leq 2^l$. We partition $\mathcal{J}$ by letting $J_l = \{j \in \mathcal{J} : \ell(j) = l\}$ for each $l \in [L]$.

Chudak and Shmoys [30] solve a similar LP without Constraint 3.15 to get an $O(\log m)$ approximation for $Q|\text{prec}, r_j|\sum_j w_j C_j$. They follow a similar framework by first giving a makespan minimizing algorithm for $Q|\text{prec}, r_j|C_{max}$ and then using it on J_l's as a subroutine to get the same result for the weighted completion time objective. We replace Graham's list scheduling for $P|\text{prec}|C_{max}$ with Chudak and Shmoys' speed-based list scheduling algorithm [30] for $Q|\text{prec}, r_j|C_{max}$ in our first step. We now briefly discuss their speed-based list scheduling algorithm, its analysis, and how to integrate it into our setup.

Their speed-based list scheduling algorithm uses a list to process the jobs greedily in the order given by the list similar to Graham's list scheduling. In addition, it uses a function f mapping each job j to a speed $f(j) \in \{\mathrm{f}_1, \mathrm{f}_2, \cdots \mathrm{f}_K\}$. As soon as a job finishes processing, all idle machines are considered in some fixed order one by one. The algorithm considers each machine with speed f_k and schedules the first available job j in the list with $f(j) = \mathrm{f}_k$ on this machine, where a job is available if all its predecessors are completed. They analyze this algorithm by considering the Load Bound of each speed group separately. They use an argument similar to the analysis of Graham's list scheduling by charging the busy time intervals on the Load Bounds and idle intervals on the Chain Bound. They show that the length of the makespan returned by this algorithm is at most the sum of the Load Bounds for each speed group and the Chain Bound:

$$\sum_{k=1}^{K}\frac{1}{m_k}\sum_{j:f(j)=\mathrm{f}_k}\frac{p_j}{f(j)} + \max_{\mathcal{C} \text{ is a chain}}\sum_{j\in\mathcal{C}}\frac{p_j}{f(j)}$$

Note that the bound given above depends heavily on job to speed assignments f used by the algorithm. Given the assignments of an optimal solution, the Load Bound of each speed group and the Chain bound will be bounded by the optimal makespan length. This would

put the makespan of the algorithm within a factor $K+1$ of the optimal solution length. Since we do not have the optimal assignments, we now describe how to get approximate job to speed assignments f by applying standard filtering techniques on the fractional solution $\{\tilde{x}_{kjt}\}$ to LP_Q. Consider a partition J_l and a job $j \in J_l$. Let $\alpha_j = \sum_{i=1}^{K}\sum_{t=1}^{l}\tilde{x}_{kjt}$ and let $\overline{x}_{kj} = \sum_{t=1}^{l}\tilde{x}_{kjt}/\alpha_j$. Note $\alpha_j \geq 1/2$ by definition of $\ell(j) = l$. Let the average processing time of a job j in LP_Q solution be $p_j^{avg} = \sum_{k=1}^{K}(p_j/\mathrm{f}_k)\overline{x}_{kj}$. We define $f(j)$ to be the speed of the maximum capacity speed group among all speeds more than half the average speed that j is assigned to in the fractional LP_Q solution.[3] Formally, let

$$f(j) = \operatorname*{argmin}_{\mathrm{f}_k:\ p_j/\mathrm{f}_k \leq 2p_j^{avg}} \frac{p_j}{\mathrm{f}_k m_k}.$$

Similar to [30], we show that the sum of the Load Bounds of all speed groups under these job to speed assignments given by f is bounded by $4K \cdot 2^l$:

Lemma 3.5.2.

$$\sum_{k=1}^{K}\frac{1}{m_k}\sum_{j\in J_l:\ f(j)=\mathrm{f}_k}\frac{p_j}{\mathrm{f}_k} \leq 4K\cdot 2^l.$$

Proof. Let $\hat{y}_{kj} = 1$ if $f(j) = \mathrm{f}_k$ and $\hat{y}_{kj} = 0$ otherwise. Given that $f(j)$ minimizes $p_j/\mathrm{f}_k m_k$ by definition, it is easy to verify that $\{\hat{y}_{kj}\}$ is an optimal solution to the following LP:

$$\begin{aligned}
\min \quad & \sum_{k=1}^{K}\frac{1}{\mathrm{f}_k m_k}\sum_{j\in J_l} p_j y_{kj} && \text{s.t.} \\
& \sum_{k=1}^{K} y_{kj} = 1 && \forall j \in [n]; \\
& y_{kj} = 0 && \forall j \in J_l, k \in [K] : \mathrm{f}_k < p_j/(2p_j^{avg}); \\
& y_{kj} \geq 0 && \forall j \in J_l, k \in [K]
\end{aligned}$$

We demonstrate a feasible fractional solution $\{\overline{y}_{kj}\}$ satisfying these constraints with objec-

3. Equivalently, since p_j is a constant in the evaluation of $f(j)$, we can define it to be the f_k that maximizes $\mathrm{f}_k m_k$.

tive value at most $4K \cdot 2^l$. Let $\overline{y}_{kj} = 0$ if $\mathrm{f}_k < p_j/(2p_j^{avg})$ and $\overline{y}_{kj} = \overline{x}_{kj} / \sum_{k:\ p_j/\mathrm{f}_k \le 2p_j^{avg}} \overline{x}_{kj}$ for $\overline{x}_{kj}$s as defined above from the solution to LP_Q. $\{\overline{y}_{kj}\}$ is clearly a feasible solution to the LP above. We have

$$\begin{aligned}
\sum_{k=1}^{K} \frac{1}{m_k} \sum_{j \in J_l:\ f(j)=\mathrm{f}_k} \frac{p_j}{\mathrm{f}_k} = \sum_{k=1}^{K} \frac{1}{m_k \mathrm{f}_k} \sum_{j \in J_l} p_j \hat{y}_{kj} &\le \sum_{k=1}^{K} \frac{1}{m_k \mathrm{f}_k} \sum_{j \in J_l} p_j \overline{y}_{kj} \\
&\le 2 \sum_{k=1}^{K} \frac{1}{m_k \mathrm{f}_k} \sum_{j \in J_l} p_j \overline{x}_{kj} \\
&\le 4 \sum_{k=1}^{K} \frac{1}{m_k \mathrm{f}_k} \sum_{j \in J_l} p_j \sum_{t=1}^{\ell(j)} \tilde{x}_{kjt} \\
&\le 4K \cdot 2^l
\end{aligned}$$

The equality is due to the definition of $\hat{y}_{kj}$'s. The first inequality follows from the fact that $\{\hat{y}_{kj}\}$ is an optimal solution and $\{\overline{y}_{kj}\}$ is a feasible solution to the same LP. The second inequality is due the definition of $\{\overline{y}_{kj}\}$ and the observation that $\sum_{k:\ p_j/\mathrm{f}_k \le 2p_j^{avg}} \overline{x}_{kj} \ge 1/2$. The third inequality holds since $\alpha_j \ge 1/2$ in the definition of $\{\overline{x}_{kj}\}$ from $\{\tilde{x}_{kj}\}$. Finally, the last inequality follows from the Constraint 3.13 of LP_Q and that we are only considering jobs from a single partition J_l and $\ell(j) = l$ for each j. □

Now we show that the total processing time of any chain in J_l under the assignments given by f is also bounded:

Lemma 3.5.3. *For any chain $\mathcal{C}$ of precedence constraints from $\prec_{J_l}$, we have $\sum_{j \in \mathcal{C}} p_j/f(j) \le 4 \cdot 2^l$.*

Proof. Since $p_j/f(j) \le 2p_j^{avg}$ by definition of f, we have

$$\begin{aligned}
p_j/f(j) \le 2p_j^{avg} &= 2 \sum_{k=1}^{K} (p_j/\mathrm{f}_k) \overline{x}_{kj} \\
&\le 4 \sum_{k=1}^{K} (p_j/\mathrm{f}_k) \sum_{t=1}^{l} \tilde{x}_{kjt}
\end{aligned}$$

Summing the terms for each job in the chain and using Constraint 3.11 repeatedly, we get

$$\sum_{j\in\mathcal{C}} \frac{p_j}{f(j)} \leq 4 \sum_{j\in\mathcal{C}} \sum_{k=1}^{K} \frac{p_j}{\mathrm{f}_k} \sum_{t=1}^{l} \tilde{x}_{kjt} \leq 4C_{j'}$$

where j' is the last job in the chain $\mathcal{C}$. Since $j' \in J_l$, we have that $l = \ell(j') \geq \ell_2(j')$. Thus $C_{j'} \leq 2^l$. □

Lemmas 3.5.2 and 3.5.3 show that the speed-based list scheduling algorithm we employ in the first step returns a schedule of length $O(K) \cdot 2^l$ on the smaller instance we get from the partition J_l.

After getting the intermediate schedule from the first step of our algorithm, we run our second step (Divide and Schedule) respecting job to machine assignments of the intermediate schedule. Note that J_{mid} in **DS** has only one job per machine because jobs in J_{mid} all run in parallel at some point in the intermediate schedule. Thus, we do not need any modification to **NFDH** subroutine to ensure the same job to machine assignments in the final schedule.

Next, we show that Resource Bound of the sub instance obtained from the partition J_l is bounded by $4S \cdot 2^l$ under the same job to speed assignments f.

Lemma 3.5.4. $\mathsf{RB}(J_l) \leq 4S \cdot 2^l$.

Proof. We can use the same properties of the filtering techniques we used to define f and finally the Constraint 3.15 to show

$$\mathsf{RB}(J_l) = \sum_{j\in J_l} \frac{p_j}{f(j)} s_j \leq 2 \sum_{j\in J_l} p_j^{avg} s_j \leq 2 \sum_{j\in J_l} \sum_{k=1}^{K} \frac{p_j}{\mathrm{f}_k} \overline{x}_{kj} s_j \leq 4 \sum_{j\in J_l} \sum_{k=1}^{K} \frac{p_j}{\mathrm{f}_k} s_j \sum_{t=1}^{l} \tilde{x}_{kjt} \leq 4S \cdot 2^l$$

□

Our second step returns a schedule of length bounded by $\mathbf{DS}(J) \leq 2\mathsf{RB}(J)/S + \mathsf{LS}_J \log |J|$ where LS_J is the length of the intermediate schedule obtained by this modified first step

explained above. Lemmas 3.5.2, 3.5.3, and 3.5.4 prove that, for any $l \in [L]$, this bound is at most $O(K \log n) \cdot 2^l$ on the sub-instance obtained from the jobs in J_l.

As we did in Section 3.4.2, we schedule the jobs in J_l for each $l \in [L]$ to start after $O(K \log n)(1+2+4\cdots+2^l)$ and complete before $O(K \log n)(1+2+4\cdots+2^l+2^{l+1})$. This again gives us a feasible schedule as in Lemma 3.4.11 and, given that $2^{l-2} \leq \tilde{C}_j$ as in the proof of Lemma 3.4.12, we have an $O(K \log n)$ approximation.

Finally using the preprocessing of Chudak and Shmoys [30], we can assume that we only have $K = \log m$ distinct speed groups. This comes with a loss of an additional constant factor on the approximation ratio of our algorithm for the minimum makespan objective. We first discard all the machines with speed less than $1/m$ times the speed of the fastest machine and then round down each remaining speed to the nearest power of $1/2$. Discarding slow machines only increases the optimal makespan by a factor of 2 because we discard no more than m machines each of which is at most as fast as $1/m$ times the speed of the fastest machine. This means the fastest machine can process the jobs of the discarded machines with a loss of factor 2 in the length of the schedule. Similarly, we only lose another factor of 2 by rounding down the speeds of the remaining machines. Since we have at most $\log m$ distinct speeds after the preprocessing, the algorithm described above on each $Q|\text{res1}, \text{prec}, r_j|C_{max}$ sub-instance is an $O(\log m \log n)$-approximation with $K = \log m$.

3.6 An Empirical Study of the Divide-and-Conquer Algorithm

The scheduling problem we study have a direct application in an emerging problem in High Perfomance Computing, namely, scheduling precedence constrained jobs (also known as"DAG scheduling") on machines sharing a "power budget". In this section, we propose a modified version of our base algorithm from Section 3.4.1 as a solution to this problem, discuss it in the context of other candidate solutions using the more common "greedy" approach, and compare its performance to these other candidate solutions in a simulation environment.

The machine scheduling problem in the HPC space is usually encountered with *configurations* corresponding to different resource/run-time (e.g. power/run-time) tradeoffs for each job. These could be software configurations for the jobs or hardware configurations for the machines running the jobs. Two configurations may not be comparable directly. Consider, for example, two hardware configurations on the same machine one with 2 active cores running at 2GHz with hyperthreading enabled and 1 available memory controller and the other with 4 active cores running at 1GHz with hyperthreading disabled and 2 available memory controllers. However, these configurations are comparable in terms of the power/run-time trade-offs they provide for a particular job. Also, note that two jobs may not have the same reaction to different configurations, i.e. different configurations may be advantageous for different jobs. Thus, on identical machines, we associate configurations with jobs, not with machines. Formally, we are given, for each job $j \in \mathcal{J}$, a set of configurations $\mathcal{C}_j$ each $c \in \mathcal{C}_j$ with a processing time $\mathrm{p}(c) \in \mathbb{Z}_{\geq 0}$ and a resource (power) requirement $\mathrm{s}(c) \in \mathbb{Z}_{\geq 0}$ for j. A feasible schedule now needs to consider the power/run-time tradeoff of different configurations, choose one $c \in \mathcal{C}_j$ for each job j, and schedule it to run for $\mathrm{p}(c)$ amount of time consuming $\mathrm{s}(c)$ amount of the available power during that time.

Note that an algorithm to minimize the makespan in the case without configurations has a single intuitive goal of maximizing the resource budget utilization at any point in the schedule. In other words, a schedule using most of the resource budget available at all times will have a makespan very close to the optimal makespan length because Resource Bound is a lower bound on the optimal makespan length. However, in the case with configurations, different configurations may have very different resource consumption and Resource Bound only applies when considered for the most resource efficient configurations for each job. Thus, in the configurable case, we have an additional goal: minimizing the resource consumption of the chosen configurations.

3.6.1 Divide-and-Conquer vs Greedy Strategies

The problem we study and the solution we offer with the divide-and-conquer algorithm is not about the intricacies of scheduling introduced in the systems level such as data locality or network delays, but it is about the essentials of precedence constrained scheduling under a power-cap. (It could also be argued that some of these additional complexities are easier to deal with in a greedy algorithm and some are easier for a divide-and-conquer algorithm.) For this reason, we compare our algorithm to greedy algorithms that form the basis of today's state-of-the-art schedulers.

One such algorithm for precedence constrained scheduling under a power cap is obtained by extending the existing precedence constrained schedulers to take the power cap into account. The list scheduling algorithm proposed initially by Graham [41] has been the essential precedence constrained scheduling algorithm. There is, in fact, a hardness of approximation result [87] stating that one cannot get a better generic algorithm for precedence constrained scheduling. In the list scheduling algorithm, the precedence constraints are extended to a total order (a list) and, as soon as a machine is available, the next job in this list is scheduled on this machine. The analysis in [41] states that an arbitrary extension to a total order is as good as any extension. However, with the introduction of the power cap and requirements, we note that this extension can have an effect on the algorithm's performance. Thus we consider two greedy variants differing in the choice of the next jobs to be scheduled among all jobs whose predecessors are all completed:

- **G1:** Always take the next job to be the one whose dependencies completed the first (i.e. the one that has been waiting in the queue the longest).

- **G2:** Take the "best fit" among the jobs whose dependencies are completed. We define the best fit to be the job whose power requirement in its most energy efficient configuration is closest to the free power budget. This definition aims to minimize the deviation from the most energy efficient configuration in the final schedule.

We let **G3** refer to a greedy algorithm obtained by extending a power-cap-aware scheduler to also obey the precedence constraints. It employs the strategy of greedily scheduling the jobs in the non-increasing order of their running time, which is previously used and analyzed in the context of strip packing, resource constrained scheduling, and malleable job scheduling [38, 89]. Since the running time of a job depends on the configuration selected for it, to be able to compare jobs in terms of their run time before scheduling them, **G3** uses their run times in their most energy efficient configuration. In order to make it also satisfy the precedence relationships, it only considers the jobs whose predecessors all have been completed.

In all three algorithms described above, the next job is scheduled on the available machine in a configuration that will use the maximum amount of the free power budget. This makes sure that all three algorithms work towards at least one of two goals stated above to achieve better performance: maximize power budget utilization. Note that **G2** also makes the choice for the next job to achieve the second goal simultaneously: maximize energy efficiency of the individual job's configurations.

The state-of-the-art in scheduling is adapting greedy scheduling to practical implementation concerns. For example, Spark is a distributed workload management system that is precedence aware, but not power-aware. Spark adapts a greedy scheduler to reduce IO overhead [90]. Cilk++—a language and runtime for multi-threaded workloads—uses a sophisticated work-stealing scheduler that distributes the scheduling load across all participating processors, while maintaining the fundamentally greedy strategy [57]. Several recent advances have dealt with the practical reality that often precise power and performance tradeoffs are not known, so they augment greedy schedulers with machine learning to estimate the power and time of each configuration and job [33, 34, 68, 70, 69, 96, 97] or to deal with jobs that take longer than expected (also known as "stragglers") [32, 92]. Even modern power-aware schedulers use greedy strategies, including both commonly deployed solutions [86] and cutting-edge research solutions that deal with serious practical concerns like unknown power/performance relationships [75], manufacturing variability across super-

computer nodes [44], and balancing the needs of independent IO- and compute-bound applications [83]. Our evaluation elides these practical details to focus on the fundamental differences between greedy strategies (as described above) and the divide-and-conquer strategy.

3.6.2 Implementation of the Divide-and-Conquer Algorithm

We note that the algorithm **DS** in Section 3.4 works on fixed configurations as opposed to making the choice for the configuration of each job. Here we describe how we incorporate configuration selection into this algorithm.

We let **DS** apply **NFDH** to J_{mid} using the most energy efficient configurations ($\forall j \in J_{mid}$, $\text{argmin}_{c \in \mathcal{C}_j} \text{p}(c)\text{s}(c)$). Since **NFDH** schedules these jobs in shelves greedily and shelves are always kept disjoint, there is still room for improvement within the individual shelves. In each shelf, we first use up all the unused power budget by choosing a shorter—higher power—configuration for the longest job. We do this incrementally because the longest job may not be the longest anymore after using a portion of the available power budget. Then, similarly, we take power away from the shortest job—making it longer—and use this freed power to make the longest job shorter until no more improvements are possible. These operations essentially try to make the jobs running in a shelf start at the same time, use almost all of the power budget all throughout their execution, and finish around the same time. If we have already matched jobs that have very similar run-times in their initial energy efficient configurations, then the deviation from these configurations is minimum and we achieve high power budget utilization with energy efficient configurations.

We note that getting the intermediate schedule on exactly m machines and finding a point *mid* exactly as described in **DS** both seem to be required only to overcome the technical challenges in proving Theorem 3.4.7. We replace these methods with more efficient ones that also reinforce the two main goals of scheduling with configurations: scheduling jobs with similar run-times (to utilize more power budget) in energy efficient configurations together.

We first let Graham's List scheduling in the first step of our algorithm use the run-times of the most energy efficient configurations for each job to obtain the intermediate schedule. Moreover, we set the number of machines in the Graham's List scheduling to infinity instead of m. Then, the J_{mid} in the second step may have arbitrarily many jobs in it and we depend on **NFDH** to limit the number of jobs running concurrently in the final schedule at m by closing a shelf not only when the resource limit is exceeded but also when there are more than m machines in the current shelf. Second, we take *mid* in **DS** to be the point maximizing $|J_{mid}|$ among several random picks within a reasonable range around the mid point of the intermediate schedule (in terms of its length). This definition is consistent with the principle above since a large J_{mid} often means better grained set of mid-jobs and more degree of freedom for the second step of the algorithm to match similar jobs together in separate shelves. The implemented **DS** in the rest of the thesis refers to this version.

We evaluate our divide-and-conquer algorithm on a mixture of real and synthetic workloads, using a high-level simulator. We compare our algorithm against greedy strategies **G1**, **G2**, and **G3** on a variety of both random and real-world precedence structures varying in size and parallelism depth. We compare the four algorithms against a theoretical lower bound on execution time coming from the Resource Bound. We show how different power caps impact the algorithms and that the divide-and-conquer algorithm performs especially well with more stringent power caps. Finally, we show that the divide-and-conquer approach utilizes more of the available power than greedy, and we demonstrate that our approach scales well on large instances containing up to 10,000 jobs.

3.6.3 Experimental Setup

In this section we describe the simulator used in our experiments, the properties of directed acyclic graphs (DAGs) corresponding to precedence constraints, and the power- and performance measurements obtained. The simulator is a Python package[4] that can be used

4. Available at: https://github.com/PowerCapDAG/PowerCapDagScheduling

Table 3.1: Description of DAGs with node count n and height h

Application	Description	n	h
synth-lg-long	random, generated	$10,000$	701
synth-lg-wide	random, generated	$10,000$	7
swift1	sleep jobs called iteratively	461	3
swift2	iterative calls to Tcl	$4,195$	3
npb-is	NPB kernel: integer Sort	43	5
npb-dc	NPB benchmark: data cube	188	7
backprop	backpropagation algorithm	79	5
kmeans	clustering algorithm	60	6

to implement and test any scheduling algorithm under precedence and power constraints. It takes a DAG corresponding to the precedence relationships among the jobs, a power cap and the number of machines as input and runs the desired algorithm. The output is the schedule produced by the available algorithms and their performance comparison.

Our algorithm is DAG-agnostic and can handle any shape and size of the DAG. To demonstrate, we chose a diverse ensemble of DAGs. We consider DAGs from real scientific applications and randomly generated DAGs (using DAGGEN [1]). We consider two simple Swift/T [91] applications with DAGs generated directly by Swift. We have also selected applications from the Rodinia benchmark suite [29] and the NAS Parallel Benchmarks (NPB) [10], whose DAGs were obtained by combining compiler-generated callgraphs using LLVM [56].

The simulator takes a set of configurations for each DAG node that correspond to real measurements of run-time and power of whole applications obtained from a single machine. The mapping of configurations to DAG structures described above is performed in a random fashion and is configurable. In total we have power and run-time measurements of 26 applications obtained from a real system. The applications are a combination of Rodinia and MineBench [72] benchmarks. Power and performance of each application were measured on a dual-socket Linux 3.2.0 system with a SuperMICRO X9DRL-iF motherboard and two Intel Xeon E5-2690 processors. Each processor has 8 cores with hyperthreading, and 16 DVFS settings. In addition, each processor has its own memory controller. Therefore, in total there are 1024 user-accessible configurations (16 cores, 2 hyperthreads, 2 memory controllers, 16

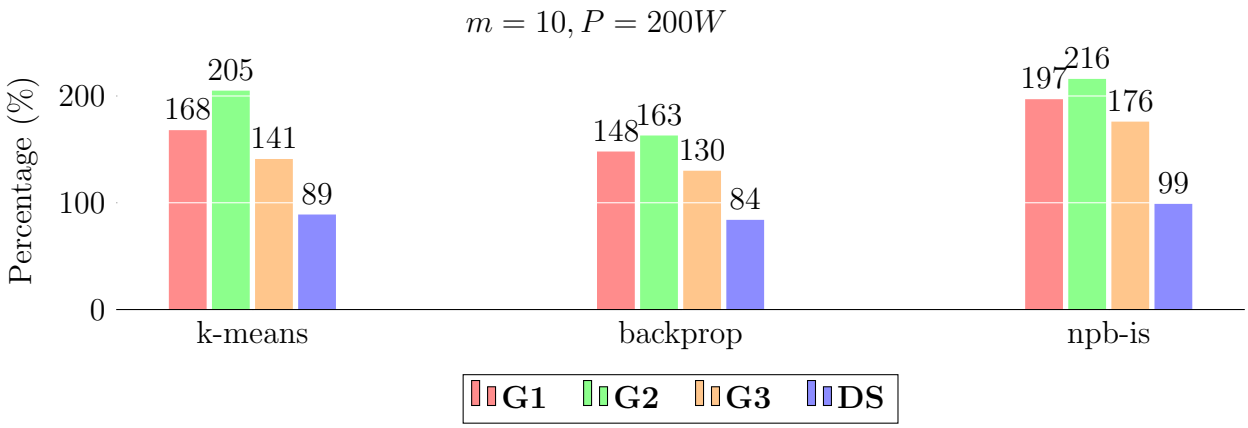


Figure 3.2: Percent overhead to theoretical lower bound on DAGs k-means, backprop, and npb-is. **DS** has significantly smaller overhead over all three greedy algorithms.

DVFS settings). This system's idle power is approximately 90 W. A user may wish to collect their own power and performance measurements and use them in our simulator instead.

3.6.4 Results

For each simulation, we obtain a theoretical lower bound on the execution time that is the Resource Bound with each job on its most energy efficient configuration. This bound can be even smaller than what is attainable by an algorithm because it does not consider the precedence constraints. Nevertheless, this lower bound on the execution time is the best that any algorithm can get. We then obtain the additional time of **DS**, **G1**, **G2**, and **G3** beyond this lower bound in terms of percentages for each simulation. We refer to this additional time as the *overhead* of the strategy compared to the lower bound. With each combination of m, P and DAG parameters, we run a hundred simulations and take the geometric mean of the percent overheads.

Figure 3.2 shows the mean percent overheads of all four algorithms with $m = 10$ machines under $P = 200W$ power cap. On the precedence constraints implied in kmeans DAG, for example, **DS** has on average 89% overhead to the theoretical lower bound whereas the best out of three greedy algorithms, **G3**, has 141%. Thus, we can consider **DS** to have a $(141 - 89)/141 = 36.9\%$ improvement over the best greedy algorithm in this setting.

Table 3.2: Percent improvement of **DS** over the best of **G1**, **G2**, and **G3** in each case with 10 and 20 machines and various power caps.

m=10	*backprop*	*kmeans*	*npb-dc*	*npb-is*	*swift1*	*swift2*	*synth-lg-long*
P=100W	60.3	60.0	67.4	64.6	63.9	72.0	32.7
P=200	35.4	36.9	36.2	43.8	40.4	46.2	13.4
P=300	5.8	18.5	9.9	15.5	4.5	3.6	5.0
P=400	3.8	3.1	-7.2	11.8	-12.2	0.0	2.3
P=500	-8.2	5.3	-15.7	-3.0	-14.3	2.4	-18.8
m=20							
P=200W	62.3	61.5	59.8	72.8	74.6	64.3	
P=400	39.4	37.8	49.0	39.2	27.7	30.0	
P=600	2.3	12.9	-8.7	8.2	10.0	-13.2	
P=800	1.9	3.4	-27.2	5.6	6.8	-20.4	
P=1000	-25.5	-10	-15.5	-9.3	-2.3	-24.7	

Table 3.2 summarizes our main results in terms of percent improvement of **DS** over the *best* greedy algorithm in each setting. When power is severely constrained ($P \leq 300$ for $m = 10$, $P \leq 600$ for $m = 20$) we find that **DS** averages 35% improvement over the best of greedy algorithms. When power constraints are loose **DS** tends to have lower performance because the problem starts to become more like precedence constrained scheduling with no power (i.e. resource) constraints for which the greedy algorithms have been known to perform very well for 50 years [41].

Different global power cap settings impact the efficacy of any scheduling algorithm under power constraints. When the power cap is too liberal, the problem reduces to a precedence scheduling problem with essentially no resource constraints. To avoid this, for each combination of graph and number of machines, we set the power cap within a range of the mean statistics of the most energy efficient configurations' power requirements because, as explained before, any such effective scheduling algorithm should aim for these configurations.[5]

5. Mean power requirement of the most energy efficient configurations in different simulations ranges from

Table 3.3: Percent overhead of **G1**|**G2**|**G3**|**DS**, respectively, on the theoretical lower bound. Lower is better. For each entry the best result is in bold and the second best is italic.

m=10	backprop	kmeans	npb-dc	npb-is	swift1	swift2	synth-lg-long
P=100W	*73*|100|78|**29**	*115*|172|128|**46**	*46*|64|*46*|**15**	149|215|*127*|**45**	*36*|46|41|**13**	26|39|*25*|**7**	60|90|*55*|**37**
P=200	148|163|*130*|**84**	168|205|*141*|**89**	62|65|*58*|**37**	197|216|*176*|**99**	*47*|50|*47*|**28**	*26*|*26*|*26*|**14**	182|280|*149*|**129**
P=300	178|193|*155*|**146**	228|255|*195*|**159**	94|111|*81*|**73**	329|323|*245*|**207**	45|*44*|*44*|**42**	29|29|*28*|**27**	293|394|*221*|**210**
P=400	201|206|*182*|**175**	227|234|*195*|**189**	*83*|88|**77**|*83*	378|397|*321*|**283**	*37*|*37*|**36**|41	*35*|*35*|**33**|**33**	280|339|*215*|**210**
P=500	319|361|**259**|*282*	372|427|*323*|**306**	*118*|129|**113**|134	304|356|**262**|*270*	*62*|*62*|**60**|70	43|43|*42*|**41**	351|429|**277**|*341*
m=20							
P=200W	224|315|*220*|**83**	252|403|*281*|**97**	*92*|136|100|**37**	*383*|491|398|**104**	*67*|92|*67*|**17**	29|39|*28*|**10**	
P=400	247|247|*213*|**129**	378|416|*328*|**204**	123|124|*104*|**53**	558|630|*492*|**299**	74|95|*65*|**47**	31|32|*30*|**21**	
P=600	420|491|*344*|**336**	471|565|*412*|**359**	180|194|**137**|*150*	508|525|*379*|**348**	85|83|*70*|**63**	**33**|**33**|**33**|*38*	
P=800	480|563|*429*|**421**	659|725|*527*|**509**	*156*|176|**134**|184	832|985|*709*|**669**	*117*|132|313|**109**	*44*|*44*|**43**|54	
P=1000	513|499|**369**|*495*	728|780|**583**|*648*	361|410|**288**|*341*	924|971|**708**|*781*	**127**|131|137|*130*	**58**|*59*|**58**|77	

Table 3.2 demonstrates the superiority of our divide-and-conquer method under strict power caps.[6] Improvements get to as high as 74.6% in the case of $m = 20, P = 200W$ on the swift1 DAG. Negative percentages indicate **DS** underperforming the best greedy algorithm in that case. **DS**'s performance over each of there greedy algorithm separately can be found in Table 3.3. A self-evident pattern in these results is the diminishing improvements of **DS** as the power cap gets bigger. In all experiments, there is a *turning point* for the power cap where **DS** performs significantly better for power caps below this point and, when the power cap is above this point, either one of the greedy algorithms starts to perform better or all four algorithms tend to perform similarly depending on the particular DAG. For example, npb-is DAG on $m = 20$ machines has this turning point somewhere between $P = 800W$ and $P = 1000W$. As mentioned earlier, precedence scheduling under power constraints starts to resemble precedence scheduling with no power constraints as the power cap increases. Since our divide-and-conquer method is particularly designed for scheduling under resource constraints, the observed diminishing improvements of **DS** as the power cap increases are exactly what is expected.

The principle of concurrently scheduling jobs with similar run-times in their most energy

$15W$ to $20W$ above idle.

6. As the DAG synth-lg-long is a "long" graph with very limited parallelism, we only report $m = 10$ for synth-lg-long.

efficient configurations gives our divide-and-conquer method an edge in better managing the trade-off between conflicting approaches of choosing energy efficient configurations and utilizing the power budget. We also note that this trade-off is biased towards the power budget utilization. Our results for energy efficiency of the configurations chosen by all four algorithms and for the power utilization attained by them support this claim. In majority of the cases where **DS** outperforms greedy algorithms, it utilizes significantly more power budget on average than these algorithms. In the combinations of machine numbers, power caps, and DAGs reported in Table 3.2, greedy algorithms waste 44.2% of the power budget on average whereas **DS**'s non-utilized power budget is only 30% of the total available budget. Still, these values may seem considerably high for both algorithms. As one would expect, both algorithms manage to utilize a much bigger portion of the power budget when the power cap is low compared to the cases when the power cap is high. On the npb-dc graph with $m = 10$ machines, for example, **G3** and **DS** fail to utilize 10.9% and 3.8% when the power budget is $P = 100W$, respectively, but these numbers get as high as 36.6% and 32.8% when it is $P = 500W$. Neither algorithm can significantly utilize a power budget set by a high power cap, because of two reasons: applications have realistic configuration sets in which arbitrarily increasing the power requirements of a job—thereby speeding up the job—is not possible, and all these DAGs have parts with insufficient parallelism to utilize the whole power budget. Moreover, the poor power utilization by both algorithms is another indicator that the problem on high power caps starts to turn into precedence scheduling with no power cap.

When **DS** cannot utilize more power than the greedy algorithms, it still has an advantage by scheduling jobs closer to their energy efficient configurations than the greedy algorithms can. On swift2 with $m = 10$ and $P = 100W$, for example, **DS** and **G3** fail to utilize 3.4% and 1.3% of the power budget, respectively, but **DS** still manages to get better final execution times by choosing configurations with total energy only 4% more than the best possible. In the same setting, **G3**, the best greedy algorithm among all three in this particular setting,

Table 3.4: The algorithm scales well to 50 and 100 machines. Percent overhead of **G1**|**G2**|**G3**|**DS**, respectively, on the theoretical lower bound.

m=50	swift1	swift2	synth-lg-wide
P=500W	132|165|*93*|**69**	*34*|46|36|**18**	*29*|43|**28**|32
P=800	*219*|450|399|**120**	*43*|47|44|**32**	**36**|48|*38*|53
P=1000	317|605|*260*|**161**	**41**|50|*42*|**41**	*43*|66|**36**|74
P=1500	*541*|927|746|**259**	**32**|*33*|**32**|53	*58*|88|**41**|123
P=2000	535|846|*515*|**270**	**54**|*55*|**54**|97	*50*|57|**42**|124
m=100			
P=500W	124|141|*82*|**60**	*31*|38|39|**14**	**29**|37|40|*33*
P=800	*215*|441|292|**114**	*40*|55|49|**27**	*48*|92|**45**|67
P=1000	312|574|*271*|**154**	*52*|85|58|**41**	*48*|86|**45**|71
P=2000	613|897|*612*|**252**	**62**|101|*66*|82	*97*|119|**66**|103
P=3000	925|1153|*916*|**349**	46|*43*|**41**|96	*180*|212|**125**|217

chooses configurations with total energy 24% more than the energy of the most energy efficient configurations. Thus, in this case **DS** is saving significant energy compared to the best greedy approach.

In addition to improvement of **DS** over the best greedy algorithm in each case presented in Table 3.2, we present our full results as each algorithms' overhead to theoretical lower bound in Table 3.3. In Table 3.4, we provide additional results in support of the scalability of our algorithm. We present the settings with $m = 50$ and $m = 100$ on the DAGs that are large enough and that have enough parallelism to benefit from the increase in the number of machines. One might consider increasing the range of power caps proportional to the number of machines m. However, a power cap set this way for $m = 100$ machines to $P = 6000W$, for example, will reduce the problem to essentially precedence constrained scheduling with no power cap at all times when the DAG cannot utilize all 100 machines. Also recent research indicates it may be more efficient overall to more severely cap large systems and make more use of parallelism than speed of individual nodes [74, 82]. We report in the range $500 - 2000W$ for 50 machines and in the range $500 - 3000W$ for 100 machines. We

observe the same behavior of **DS** having diminishing improvements as we increase the power cap. We note that the size of any particular DAG does not change in our simulations as we increase m from 10 to 100 and P from $100W$ to $3000W$; i.e., we are simulating strong scaling results.

REFERENCES

[1] DAGGEN: A Synthetic Task Graph Generator, 2013.

[2] Karen Aardal, Pieter L. van den Berg, Dion Gijswijt, and Shanfei Li. Approximation algorithms for hard capacitated k-facility location problems. *European Journal of Operational Research*, 242(2):358 – 368, 2015.

[3] Susanne Albers. Algorithms for dynamic speed scaling. In *STACS*, pages 1–11, 2011.

[4] Susanne Albers and Antonios Antoniadis. Race to idle: new algorithms for speed scaling with a sleep state. In *SODA*, 2012.

[5] Hyung-Chan An, Mohit Singh, and Ola Svensson. LP-based algorithms for capacitated facility location. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014*.

[6] Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for euclidean k-medians and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 106–113, New York, NY, USA, 1998. ACM.

[7] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 21–29, New York, NY, USA, 2001. ACM.

[8] John Augustine, Sudarshan Banerjee, and Sandy Irani. Strip packing with precedence constraints and strip packing with release times. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, pages 180–189, New York, NY, USA, 2006. ACM.

[9] Hakan Aydi, Pedro Mejía-Alvarez, Daniel Mossé, and Rami Melhem. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *RTSS*, 2001.

[10] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The nas parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[11] Peter E. Bailey, Aniruddha Marathe, David K. Lowenthal, Barry Rountree, and Martin Schulz. Finding the limits of power-constrained application performance. In *SC*, 2015.

[12] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. *Algorithmica*, 60(4), 2011.

[13] Nikhil Bansal, Ho-Leung Chan, Tak Wah Lam, and Lap-Kei Lee. Scheduling for speed bounded processors. In *ICALP*, 2008.

[14] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. *ACM Transactions on Algorithms*, 9(2), 2013.

[15] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 161–168, New York, NY, USA, 1998. ACM.

[16] Abbas Bazzi and Ashkan Norouzi-Fard. Towards tight lower bounds for scheduling problems. In *Proceedings of the 23rd Annual European Symposium on Algorithms, ESA'15*, volume 9294, page 118. Springer, 2015.

[17] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, Peter Kogge, R. Stanley Williams, and Katherine Yelick. Exascale computing study: Technology challenges in achieving exascale systems peter kogge, editor & study lead, 2008.

[18] Enrico Bini, Giorgio C. Buttazzo, and Giuseppe Lipari. Minimizing cpu energy in real-time systems with discrete speed management. *ACM Trans. Embedded Comput. Syst.*, 8(4), 2009.

[19] David P. Bunde. Power-aware scheduling for makespan and flow. In *SPAA*, 2006.

[20] Jarosław Byrka, Krzysztof Fleszar, Bartosz Rybicki, and Joachim Spoerhase. Bi-factor approximation algorithms for hard capacitated k-median problems. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*.

[21] Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*.

[22] Jarosław Byrka, Bartosz Rybicki, and Sumedha Uniyal. An approximation algorithm for uniform capacitated k-median problem with $1 + \epsilon$ capacity violation, 2015. arXiv:1511.07494.

[23] Jarosław Byrka, Bartosz Rybicki, and Sumedha Uniyal. An approximation algorithm for uniform capacitated k-median problem with 1+\epsilon capacity violation. In *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO*, pages 262–274, 2016.

[24] Robert D. Carr, Lisa K. Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 106–115, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[25] Ho-Leung Chan, Joseph Wun-Tat Chan, Tak Wah Lam, Lap-Kei Lee, Kin-Sum Mak, and Prudence W. H. Wong. Optimizing throughput and energy in online deadline scheduling. *ACM Transactions on Algorithms*, 6(1), 2009.

[26] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *In Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 378–388, 1999.

[27] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem (extended abstract). In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 1–10, New York, NY, USA, 1999. ACM.

[28] Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. Rounding via trees: Deterministic approximation algorithms for group steiner trees and k-median. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, STOC '98, pages 114–123, New York, NY, USA, 1998. ACM.

[29] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*, 2009.

[30] Fabián A. Chudak and David B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, pages 581–590, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[31] Julia Chuzhoy and Yuval Rabani. Approximating k-median with non-uniform capacities. In *SODA '05*, pages 952–958, 2005.

[32] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI'04: Proc. of the 6th conf. on Opearting Systems Design & Implementation*, 2004.

[33] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *ASPLOS*, 2013.

[34] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *ASPLOS*, 2014.

[35] Gökalp Demirci, Henry Hoffmann, and David H. K. Kim. Approximation Algorithms for Scheduling with Resource and Precedence Constraints. In *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *LIPIcs*, pages 25:1–25:14, 2018.

[36] Gökalp Demirci and Shi Li. Constant Approximation for Capacitated k-Median with (1+epsilon)-Capacity Violation. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 73:1–73:14, 2016.

[37] Gökalp Demirci, Ivana Marincic, and Henry Hoffmann. A Divide and Conquer Algorithm for DAG Scheduling Under Power Constraints. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, pages 36:1–36:12, 2018.

[38] Jr. E. G. Coffman, M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

[39] M. R. Garey and R. L. Grahams. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4:187–200, 1975.

[40] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287 – 326, 1979. Discrete Optimization II.

[41] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell Labs Technical Journal*, 45(9):1563–1581, 1966.

[42] Sudipto Guha. *Approximation Algorithms for Facility Location Problems*. PhD thesis, Stanford, CA, USA, 2000.

[43] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.

[44] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, Masaaki Kondo, and Ikuo Miyoshi. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *SC*, 2015.

[45] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Trans. Algorithms*, 3(4), November 2007.

[46] K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 731–740, New York, NY, USA, 2002. ACM.

[47] K Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

[48] Klaus Jansen, Marten Maack, and Malin Rau. Approximation schemes for machine scheduling with resource (in-)dependent processing times. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1526–1542, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.

[49] Klaus Jansen and Hu Zhang. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Trans. Algorithms*, 2(3):416–434, July 2006.

[50] Klaus Jansen and Hu Zhang. Scheduling malleable tasks with precedence constraints. *Journal of Computer and System Sciences*, 78(1):245 – 259, 2012. JCSS Knowledge Representation and Reasoning.

[51] N. Kappiah, V. W. Freeh, and D. K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. In *SC*, 2005.

[52] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. ii: The p-medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560, 1979.

[53] David H. K. Kim, Connor Imes, and Henry Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *CPSNA*, 2015.

[54] Stavros G. Kolliopoulos and Satish Rao. A nearly linear-time approximation scheme for the euclidean k-median problem. *SIAM J. Comput.*, 37(3):757–782, June 2007.

[55] Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 1–10, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[56] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Code Generation and Optimization, 2004. CGO 2004. International Symposium on*, pages 75–86. IEEE, 2004.

[57] C. E. Leiserson. The cilk++ concurrency platform. In *2009 46th ACM/IEEE Design Automation Conference*, pages 522–527, July 2009.

[58] Renaud Lepère, Denis Trystram, and Gerhard J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. In *Proceedings of the 9th Annual European Symposium on Algorithms*, ESA '01, pages 146–157, 2001.

[59] Shanfei Li. An improved approximation algorithm for the hard uniform capacitated k-median problem. In *APPROX '14/RANDOM '14: Proceedings of the 17th International Workshop on Combinatorial Optimization Problems and the 18th International Workshop on Randomization and Computation*, APPROX '14/RANDOM '14, 2014.

[60] Shi Li. On uniform capacitated k-median beyond the natural LP relaxation. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*.

[61] Shi Li. Approximating capacitated k-median with $(1 + \epsilon)k$ open facilities. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 786–796, 2016.

[62] Shi Li. Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. In *IEEE 57th Annual Symposium on Foundations of Computer Science*, FOCS '17, 2017.

[63] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 901–910, New York, NY, USA, 2013. ACM.

[64] J. Lin and J. S. Vitter. ϵ-approximations with minimum packing constraint violation (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC), Victoria, British Columbia, Canada*, pages 771–782, 1992.

[65] Jyh-Han Lin and Jeffrey Scott Vitter. Approximation algorithms for geometric median problems. *Inf. Process. Lett.*, 44(5):245–249, December 1992.

[66] Aniruddha Marathe, Peter E. Bailey, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. A run-time system for power-constrained hpc applications. In *ISC*, 2015.

[67] Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984.

[68] N. Mishra, J. D. Lafferty, and H. Hoffmann. Esp: A machine learning approach to predicting application interference. In *ICAC*, 2017.

[69] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. CALOREE: Learning Control for Predictable Latency and Low Energy. In *ASPLOS*, 2018.

[70] Nikita Mishra, Huazhe Zhang, John D Lafferty, and Henry Hoffmann. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In *ASPLOS*, 2015.

[71] Alix Munier, Maurice Queyranne, and Andreas Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In *Integer Programming and Combinatorial Optimization*, IPCO '98, 1998.

[72] Ramanathan Narayanan, Berkin Özisikyilmaz, Joseph Zambreno, Gokhan Memik, and Alok Choudhary. Minebench: A benchmark suite for data mining workloads. In *Workload Characterization, 2006 IEEE International Symposium on*, pages 182–188. IEEE, 2006.

[73] Martin Niemeier and Andreas Wiese. Scheduling with an orthogonal resource constraint. In *10th Workshop on Approximation and Online Algorithms (WAOA2012)*, number EPFL-CONF-181146, 2012.

[74] Tapasya Patki, David K. Lowenthal, Anjana Sasidharan, Matthias Maiterth, Barry L. Rountree, Martin Schulz, and Bronis R. de Supinski. Practical resource management in power-constrained, high performance computing. In *HPDC*, 2015.

[75] Tapasya Patki, David K. Lowenthal, Anjana Sasidharan, Matthias Maiterth, Barry L. Rountree, Martin Schulz, and Bronis R. de Supinski. Practical resource management in power-constrained, high performance computing. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 121–132, New York, NY, USA, 2015. ACM.

[76] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*, 2001.

[77] Kirk Pruhs, Rob van Stee, and Patchrawat Uthaisombut. Speed scaling of tasks with precedence constraints. In *WAOA*, 2006.

[78] Maurice Queyranne and Andreas S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *SIAM J. Comput.*, 35(5):1241–1253, May 2006.

[79] James Reinders. *Intel Threading Building Blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition, 2007.

[80] Vivek Sarkar, Saman Amarasinghe, Dan Campbell, William Carlson, Andrew Chien, William Dally, Elmootazbellah Elnohazy, Mary Hall, Robert Harrison, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Charles Koelbel, David Koester, Peter Kogge, John Levesque, Daniel Reed, Robert Schreiber, Mark Richards, Al Scarpelli, John Shalf, Allan Snavely, and Thomas Sterling. Exascale software study: Software challenges in extreme scale systems, 2009. DARPA IPTO Study Report for William Harrod.

[81] Vivek Sarkar, Saman Amarasinghe, Dan Campbell, William Carlson, Andrew Chien, William Dally, Elmootazbellah Elnohazy, Mary Hall, Robert Harrison, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Charles Koelbel, David Koester, Peter Kogge, John Levesque, Daniel Reed, Robert Schreiber, Mark Richards, Al Scarpelli, John Shalf, Allan Snavely, and Thomas Sterling. Exascale software study: Software challenges in extreme scale systems, 2009. DARPA IPTO Study Report for William Harrod.

[82] O. Sarood, A. Langer, L. Kale, B. Rountree, and B. de Supinski. Optimizing power allocation to cpu and memory subsystems in overprovisioned hpc systems. In *CLUSTER*, 2013.

[83] L. Savoie, D. K. Lowenthal, B. R. d. Supinski, T. Islam, K. Mohror, B. Rountree, and M. Schulz. I/o aware power shifting. In *IPDPS*, May 2016.

[84] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *VECPAR*, 2011.

[85] D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems (extended abstract). In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274, New York, NY, USA, 1997. ACM.

[86] SLURM. The slurm workload manager. Online document, `https://slurm.schedmd.com/`.

[87] Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 745–754, New York, NY, USA, 2010. ACM.

[88] ExaOSR Team. Key challenges for exascale os/r. Online document, `https://collab.mcs.anl.gov/display/exaosr/Challengesl`.

[89] John Turek, Joel L. Wolf, and Philip S. Yu. Approximate algorithms scheduling parallelizable tasks. In *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '92, pages 323–332, 1992.

[90] Shivaram Venkataraman, Aurojit Panda, Ganesh Ananthanarayanan, Michael J. Franklin, and Ion Stoica. The power of choice in data-aware cluster scheduling. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.*, 2014.

[91] Justin M Wozniak, Timothy G Armstrong, Mark Wilde, Daniel S Katz, Ewing Lusk, and Ian T Foster. Swift/t: large-scale application composition via distributed-memory dataflow processing. In *CCGrid*, 2013.

[92] Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, and Randy Katz. Multi-task learning for straggler avoiding predictive job scheduling. *Journal of Machine Learning Research*, 17(106):1–37, 2016.

[93] F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *FOCS*, 1995.

[94] Heechul Yun, Po-Liang Wu, Anshu Arya, Cheolgi Kim, Tarek F. Abdelzaher, and Lui Sha. System-wide energy optimization for multiple dvs components and real-time tasks. *Real-Time Systems*, 47(5), 2011.

[95] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010*, 2010.

[96] Huazhe Zhang and Henry Hoffmann. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. In *ASPLOS*, 2016.

[97] Huazhe Zhang and Henry Hoffmann. Performance & energy tradeoffs for dependent distributed applications under system-wide power caps. In *ICPP*, 2018.