Santa Claus, Machine Scheduling and Bipartite Hypergraphs

by

Andrew Jay

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics

Waterloo, Ontario, Canada, 2019

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Here we discuss two related discrete optimization problems, a prominent problem in scheduling theory, makespan minimization on unrelated parallel machines, and the other a fair allocation problem, the Santa Claus problem. In each case the objective is to make the least well off participant as well off as possible, and in each case we have complexity results that bound how close we may estimate optimal values of worst-case instances in polytime. We explore some of the techniques that have been used in obtaining approximation algorithms or optimal value guarantees for these problems, as well as those involved in getting hardness results, emphasizing the relationships between the problems. A framework for decisional variants of approximation and optimal value estimation for optimization problems is introduced to clarify the discussion.

Also discussed are bipartite hypergraphs, which correspond naturally to these problems, including a discussion of Haxell's Theorem for bipartite hypergraphs. Conditions for edge covering in bipartite hypergraphs are introduced and their implications investigated. The conditions are motivated by analogy to Haxell's Theorem and from generalizing conditions that arose from bipartite hypergraphs associated with machine scheduling.

Acknowledgements

I am grateful to the University of Waterloo for providing a stimulating experience during my Master's studies and for supporting my research.

Special thanks goes to my advisor Penny Haxell, whose guidance and commentary has been of immeasurable benefit to this work.

Dedication

I dedicate this work to the entire Jay family (and most especially my parents), whose support and enthusiasm for my mathematical endeavours has been overwhelming, and in the memory of my maternal grandparents.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Outlining the Thesis and its Contributions

This thesis discusses two closely related problems in discrete optimization, the *min makespan scheduling problem on unrelated machines* and the *max-min allocation problem*. These problems are each computationally complex, indeed NP-hard, to solve exactly. So at issue here is how accurately they may be efficiently approximated. Improving the long-standing current best factor for approximation of this min makespan scheduling problem is one of the most prominent open problems in scheduling theory (see for instance, a discussion of some open problems by Schuurman and Woeginger [27]). The max-min allocation problem is closely related and there has been much overlap in the techniques applied to the study of each problem. In this work we aim to summarize some of the major results concerning the polytime approximability for these problems and to give a sense of some of the techniques involved in the proofs, by going over a selection of the results in detail. A linear program relaxation, known as the Configuration LP (CLP), will be a central tool for much of the work discussed here, and we present a section (in Chapter 3) which elaborates on the details of its application to these problems (the essentials of which were briefly sketched in the original paper by Bansal and Sviridenko introducing the CLP [5]).

Many of the proofs given in this thesis for results in the literature are given in a somewhat different setting, being reworked or expanded upon for clarity and unity with the rest of the presentation. There is an emphasis on examining the similarity and differences between the two problems, showing common methods and discussing when results obtained for one fail to fruitfully carry over to the other. Another aim of this work is to explore some of the underlying ideas in these problems as they relate to *bipartite hypergraphs*.

Original Contributions:

A general framework for discussing decisional variants of approximation and optimal value estimation for optimization problems is introduced in this introductory chapter, to clarify the distinction between polytime estimation and approximability (a distinction discussed in an article by Feige [13]). The framework introduced will expand on the notion of a relaxed decision procedure already present in the literature [23] and relies upon the basic application of binary search to approximation and estimation problems.

In Chapter 4, we give an original proof for polytime estimation in a case of the max-min allocation problem, based on a result (due to Haxell [15]) concerning *independent transversals* in vertex partitioned graphs. An analogue to a matroid version of the allocation problem (introduced by Davies, Rothvoss and Zhang [11]) is introduced for the scheduling problem in Chapter 6.

The main original contribution of this thesis is in the final Chapter 8, where novel conditions on bipartite hypergraphs are introduced and explored, which we relate to a condition arising from a relaxation of the min makespan scheduling problem. We determine some consequences of these conditions, while more questions are raised to perhaps spur future research along these lines, and to close the gaps presented.

Other Contents of this Thesis:

Chapter 2 delves into the min makespan machine scheduling problem by discussing the seminal work of Lenstra, Shmoys and Tardos [23], which provides the first (and best known) constant factor approximation algorithm. In this chapter we also introduce the hardness results for our two problems.

Chapter 3 introduces the CLP relaxation for the Santa Claus problem (from the work of Bansal and Sviridenko [5]), and develops the needed results for application of the CLP to the Santa Claus and machine scheduling problems throughout this work.

Chapter 4 formulates our problems in terms of bipartite hypergraphs and discusses versions of Haxell's Theorem [15].

Chapter 5 goes over the best known result for the *restricted* case of estimation for the min makespan machine scheduling problem. First we give an introduction to the ideas in a special case that we flesh out to prepare for the main result.

Chapter 6 discusses a matroid variant of the Santa Claus problem and its application to the approximation of the usual form of the problem.

Chapter 7 deals with the graph balancing cases of these problems and some of the provably optimal approximation factors known in the literature.

Chapter 8 deals entirely with the new material mentioned, motivating it from consideration of some linear programs associated to our problems.

1.2 Introducing the Problems

We begin by formally introducing the two optimization problems that we are concerned with. The min makespan scheduling problem is frequently viewed in the context of optimal job scheduling on unrelated parallel machines as follows [23]. Suppose we have a set M of machines and a set J of jobs we wish to perform. Given times $p_{ij} \in \mathbb{R}_{>0} \cup \{\infty\}$ for machine i to perform job j we seek a job assignment function $\sigma : J \to M$ that minimizes objective $\text{MakeSpan}(\sigma) := \max_{i \in M} \sum_{j \in \sigma^{-1}(i)} p_{ij}$. The makespan here is the maximum time taken by any machine to process all assigned jobs. Jobs and machines are then a natural use of terminology, since in those terms this optimization problem is the task of scheduling

jobs on machines so that the total time from start to finish of all jobs is minimized, clearly a practical quantity to optimize. From this point on the phrases *job scheduling problem* or alternatively *machine scheduling problem*, will both refer to this min makespan problem setup. Also all problem instances will be assumed to be feasible for a finite makespan, by insisting that each job can be performed in finite time by some machine, and job assignments with an infinite makespan will be considered infeasible.

The *restricted* assignment version of this problem occurs when we have jobs with fixed performance time $p_j \in \mathbb{R}_{>0}$ for each $j \in J$, but where each job is only assignable to some subset $\Gamma(j) \subset M$ of the machines. This can be seen as a special case of the general problem taking each $p_{ij} \in \{p_j, \infty\}$. In this case performance times p_j are also referred to as *job sizes*.

The max-min allocation problem has become to be known as the *Santa Claus problem* [5]. The Santa Claus fair allocation problem gets it name from thinking of it as the task of distributing presents among children so as to make the least happy child as happy as possible, so in some sense to distribute as fairly as possible. To distinguish the two problems we are considering, the terminology used here is allocating bundles of resources to players, instead of assigning jobs to machines.

Let R be the set of resources and P the set of players, where resource j has value to player i of $v_{ij} \in \mathbb{R}_{\geq 0}$. The Santa Claus problem is to find disjoint sets/bundles of resources $\{S_i : i \in P\}$ to assign players so that objective $\min_{i \in P} \sum_{j \in S_i} v_{ij}$ is maximized. The indexed set of bundles $\{S_i : i \in P\}$ for a feasible solution forms a partition for a subset of the resources R. The optimal objective value for this problem is called the max-min. In a more general formulation [6] players are allowed to value sets of resources arbitrarily, rather than linearly based on valuations of the individual resources, but we shall not consider this here.

The restricted version of the problem occurs when resources have inherent values $v_j \in \mathbb{R}_{>0}$ for all $j \in R$ and are assignable to some subset $\Gamma(j) \subset P$ of the players. We are then restricted to choosing bundles for each player consisting entirely of resources assignable to the given player. This again may be viewed as a special case taking $v_{ij} := 0$ if and only if $i \notin \Gamma(j)$ and otherwise setting $v_{ij} := v_j$ to the inherent value for all $i \in P$ and $j \in R$.

The performance times or resource values for these problems are assumed to be given as input in binary with a finite length binary string for numerator and denominator (so a 0 in denominator can indicate infinite job performance time). The input size of these two problems is then the total bit-length of the matrix storing all p_{ij}'s/v_{ij}'s. So this assumes that all (non-infinite) values are rational, and hence integer up to some scaling factor.

We can also alternatively formulate the machine scheduling problem in terms of allocating bundles of jobs $\{S_i : i \in M\}$ to machines, instead of finding a function assigning individual jobs to machines. In this setting we are required to have all jobs contained in the union of the bundles, as we must schedule all jobs, and the objective is to minimize the makespan, the greatest time any machine takes to complete all the jobs in its assigned bundle. We may relax the requirement of disjointness, as there is no harm in a job being done multiple times and this will not affect the optimal makespan.

These problems are then quite similar in that they each seek a way to allocate bundles of items that produces an outcome which is least bad for the worst off participant, the only difference being whether the items in the bundles are goods to be distributed or onerous tasks that must be completed. Put more colourfully, it is a matter of whether we are Santa handing out presents, or Scrooge assigning overtime work.

Given the similarity, it is perhaps unsurprising that many of the ideas and techniques developed in studying one problem have carried over successfully to the other, however there are fundamental differences which make some techniques and results not translate automatically between them. In particular one notable difference to the restricted cases is that being off of optimal by misallocating a single job (changing from an optimal solution by putting one job in a different bundle) cannot be too bad for the machine scheduling, where it at most doubles the makespan, since the job was already being done by some machine. Meanwhile, in the Santa Claus problem a single resource may be in a bundle for an optimal solution, so taking it away completely ruins the solution's min allocation. This and issues related to it, have made it useful in the Santa Claus problem to separate out the resources by whether they are large in value or small, and this has consequently constrained much of the work to the restricted case, wherein this division of resources can be made independent of the player.

1.3 Some Notes on Complexity

We briefly describe some basic concepts in complexity theory needed to understand the results we will discuss. An algorithm runs in *polytime* with respect to n (a function of input most frequently taken as the input size) when there exists a polynomial in n which bounds the number of elementary operations (as modelled on a *Turing machine*) performed to execute the algorithm on any input. We say an algorithm runs in polytime when it runs in polytime with respect to the size of input (i.e. the number bits needed to encode it).

The notion of a polynomial time algorithm is understood to be a useful theoretical tool to help capture the practical notion of the efficiency or feasibility of running an algorithm (see a reference text on complexity by Arora and Barak [3]). A *decision problem* is one which has possible input strings each with a "Yes" or "No" answer. A decision problem is solved by an algorithm which always accurately outputs "Yes" or "No" to match the answer on all input. The complexity class of polytime solvable decision problems is denoted by P.

The complexity class NP consists of those decision problems that have a polytime algorithm $\mathcal{A}$ and a polynomial bound p which satisfy the following rule: for every input x (with size $|x|$) of the decision problem, there exists a *witness* w with size at most $p(|x|)$ so that $\mathcal{A}$ running on (x, w) outputs "Yes" if and only if x has (correct) answer "Yes" [3] (so when x has answer "No", there is no w of size at most $p(|x|)$ where $\mathcal{A}$ running on (x, w) will output "Yes"). The idea is that the algorithm $\mathcal{A}$ is able to verify some proof w for x being a "Yes" answer input in polytime.

Intuitively the class P consists of those decision problems that can be solved efficiently,

while NP consists of the decision problems whose solutions can be verified efficiently. Problems with complexity P are also NP, while the question of whether NP problems need be in P (the P=NP or P≠NP question) is the central open problem of complexity theory [3]. NP-hard problems are those problems $\mathcal{P}$, which if a solver for $\mathcal{P}$ is given as a polytime operation oracle (in addition to standard elementary operations), it may be used to solve any problem in the complexity class NP in polytime [3]. So NP-hard problems are at least as hard to solve in polytime as any in NP, indeed if any NP-hard problem is polytime solvable then P=NP. NP-complete decision problems are those in both NP and NP-hard, thus constituting the class of the equally "hardest" NP problems. The first decision problems to be verified NP-complete were versions of the boolean satisfiability problem by Cook [10] and Levin [24] independently. Later Karp produced a list of 21 NP-complete problems [21] and there are now thousands of different decision problems known to be NP-complete [30]. All of these NP-complete problems have resisted any attempt to be solved in polytime. Indeed it seems natural that it ought to be harder in some cases to produce a solution than to verify one, so it is widely assumed that P≠NP. For practical purposes the assumption need not even hold, it is enough to say that we have no (and will not soon have) such algorithms for solving NP-complete problems. So in terms of the practical difficulty of computation, showing a problem to be NP-hard amounts to showing its infeasibility (on large input).

Finding the optimal value of a solution to either of these two optimization problems for a general instance of the problem (in even the restricted cases) is NP-hard, as we shall see from much stronger results than this discussed later. Given the (presumed on the basis of P≠NP) barrier to efficient solutions presented by the NP-hardness of these problems, expectations for algorithmic progress must be tempered in some way. One natural aspect of the problem's complexity to study then, is determining how well in polytime we can produce approximate solutions to the problem, or even just estimate the optimal value (without producing a solution). From this point on, an *approximation algorithm* will refer only to polytime algorithms, we shall not focus on positive results concerning other complexity classes.

We shall pay particular focus to constant factor *p-approximation algorithms*, meaning an algorithm running in polytime which produces a solution within a multiplicative factor p of optimal (see this text [30] by Williamson and Shmoys for a reference on approximation algorithms). For minimization p-approximation involves finding a solution with objective at most $p*OPT$ and for maximization finding a solution with objective at least $\frac{1}{p}OPT$. We will also consider *estimation algorithms* running in polytime to determine within a constant factor p the optimal value for a problem instance, that is an algorithm determining upper and lower bounds within a factor p between which the optimal value is guaranteed to lie.

Also of interest shall be hardness results which bound how well in polytime we may estimate the optimal value of a problem instance, by showing estimating within some factor to be NP-hard (hardness results obtained by known methods always apply to estimating the optimal value in addition to the approximation factor [13]). The hardness results explored here are obtained by reduction from known NP-complete problems. A measure of theoretical progress in approximation algorithms for these types of problems is how close the hardness bounds can be made to the best known approximation algorithms

(and estimates for optimal value), and in an ideal case giving us a provably best factor approximation algorithm, provisional on P≠NP. These ideal cases have only been realized in very special cases of these two problems [17, 7], with jobs/resources assignable to at most 2 machines/players, that relate to *graph balancing* problems where one attempts to orient weighted edges so as to in some sense balance the total incoming weight to each of the vertices. A discussion of these ideal cases will be included in Chapter 7 on graph balancing, while exploring the current best results for more general cases will occupy much of this work.

1.4 Relaxed Decision Procedures

The notion of a *relaxed decision procedure* was used, in a 1990 paper due to Lenstra, Shmoys and Tardos [23], to obtain an approximation algorithm for the unrelated machine scheduling problem. Related ideas about dual approximation were introduced by Hochbaum and Shmoys in a 1987 paper on approximation algorithms and identical machine scheduling [16]. We shall adapt this notion to our purposes by drawing a distinction between two different types of relaxed decision procedures, one of which will correspond to producing approximate solutions and the other estimates for the optimal value. Firstly we clarify our meaning of an *optimization problem* for the purposes of this thesis, which should match the informal notion. We then introduce a definition for *searchable optimization problems*, which restricts *objective values* to be non-negative integers and adds a few other technical restrictions.

Definition 1.1. An *optimization problem* $\mathcal{O}$ consists of a class of *problem instances* and a Boolean variable indicating whether it is a *minimization* or *maximization* problem. Problem instances have an associated set of *feasible solutions* and each feasible solution has a real *objective value*.

The *optimal value* for $\mathbf{P}$ an instance of a minimization/maximization problem is the least/greatest (or infimum/supremum) objective value among the feasible solutions for $\mathbf{P}$, and is undefined for problem instances without any feasible solutions. We say that minimization/maximization problem instance $\mathbf{P}$ is *feasible for target* T when there exists a feasible solution with objective value at most/least T.

Definition 1.2. A *searchable optimization problem* $\mathcal{O}$ is an optimization problem, which satisfies the following conditions:
1. Objective values are all non-negative integers.
2. For every problem instance $\mathbf{P}$ of $\mathcal{O}$, the set of feasible solutions is non-empty, and the optimal value is finite.
3. There exists a polynomial q such that for every problem instance $\mathbf{P}$ of $\mathcal{O}$ with input size n, we have that $q(n)$ bounds the bit-length of the optimal value of $\mathbf{P}$ in binary representation.

These assumptions are put here so that a polytime algorithm can search through potential optimal values and the feasibility condition is here for it to be meaningful to ap-

proximate the optimal solution. Also note that the polynomial q in the above definition may be assumed to have integer coefficients.

We now introduce our two different notions of relaxed decision procedures for optimization problems. We define them separately for minimization and maximization problems.

Definition 1.3. Let $\mathcal{O}$ be a minimization problem and $p > 1$. A *type A p-relaxed decision procedure* for $\mathcal{O}$ is any algorithm $\mathcal{A}$ which takes as input an instance $\mathbf{P}$ of $\mathcal{O}$ together with a target $T > 0$, and has output satisfying the following rules:
1. $\mathcal{A}$ outputs either "No", or "Almost" together with a feasible solution for $\mathbf{P}$ with objective value at most pT.
2. If $\mathcal{A}$ outputs "No" then every feasible solution to $\mathbf{P}$ has objective value greater than T.

A *type B p-relaxed decision procedure* for $\mathcal{O}$ is an algorithm $\mathcal{B}$, which takes the same input ($\mathbf{P}$, T) and has output satisfying the following rules:
1. $\mathcal{B}$ outputs either "No", or "Almost".
2. If $\mathcal{B}$ outputs "No" then every feasible solution to $\mathbf{P}$ has objective value greater than T.
3. If $\mathcal{B}$ outputs "Almost" then there exists a feasible solution to $\mathbf{P}$ with objective value at most pT.

Definition 1.4. Let $\mathcal{O}$ be a maximization problem and $p > 1$. A *type A p-relaxed decision procedure* for $\mathcal{O}$ is any algorithm $\mathcal{A}$ which takes as input an instance $\mathbf{P}$ of $\mathcal{O}$ together with a target $T > 0$, and has output satisfying the following rules:
1. $\mathcal{A}$ outputs either "No", or "Almost" together with a feasible solution for $\mathbf{P}$ with objective value at least $\frac{T}{p}$.
2. If $\mathcal{A}$ outputs "No" then every feasible solution to $\mathbf{P}$ has objective value less than T.

A *type B p-relaxed decision procedure* for $\mathcal{O}$ is an algorithm $\mathcal{B}$, which takes the same input ($\mathbf{P}$, T) and has output satisfying the following rules:
1. $\mathcal{B}$ outputs either "No", or "Almost".
2. If $\mathcal{B}$ outputs "No" then every feasible solution to $\mathbf{P}$ has objective value less than T.
3. If $\mathcal{B}$ outputs "Almost" then there exists a feasible solution to $\mathbf{P}$ with objective value at least $\frac{T}{p}$.

Note that a type A p-relaxed decision procedure can also be made to act as a type B p-relaxed decision procedure, by omitting its output of solutions and just having it return "Almost" in those cases.

For subsequent proofs concerning the implications of a searchable optimization problem having a p-relaxed decision procedure (of type A or B), in order to avoid repetition we will assume that $\mathcal{O}$ is a maximization problem when convenient. Firstly, we show that being able to efficiently produce solutions approximately meeting a feasible target, is enough to yield a polytime type A relaxed decision procedure.

Proposition 1.5. *Let $\mathcal{O}$ be an optimization problem such that the feasibility and objective value of a candidate solution to any problem instance may be checked in polytime. Suppose $\mathcal{A}$ is an algorithm which takes as input a problem instance $\mathbf{P}$ of $\mathcal{O}$ and $T > 0$, such that when $\mathbf{P}$ is feasible for target T, it outputs a feasible solution with objective value within a factor of p of T and runs in polytime. Then $\mathcal{O}$ has a polytime type A p-relaxed decision procedure.*

Proof. Let $t(n)$ be the polynomial bound on runtime/operations of $\mathcal{A}$ running on input feasible for the target. Consider the following algorithm $\mathcal{A}'$ based on $\mathcal{A}$, which runs as follows on input ($\mathbf{P}$,T):
1. Run $\mathcal{A}$ on ($\mathbf{P}$,T) until its termination or until runtime $t(n)$ has been exceeded (for n the size of input ($\mathbf{P}$,T)).
2. If $\mathcal{A}$ has outputted a candidate solution x, we check its feasibility and objective value in polytime. If x is feasible and has objective value within a factor of p of T, we output (x,"Almost").
3. Otherwise, when $\mathcal{A}$ has failed to produce a feasible solution x with objective within a factor of p of T, we output "No".

Clearly this algorithm $\mathcal{A}'$ runs in polytime. A "No" answer from $\mathcal{A}'$ shows that $\mathcal{A}$ fails on the same input to produce a solution x as desired, and therefore certifies that T was not a feasible target for $\mathbf{P}$. The "Almost" response is accompanied by a feasible solution with objective within a factor of p of the target. Therefore $\mathcal{A}'$ is a polytime type A p-relaxed decision procedure for $\mathcal{O}$. □

A version of the following result for obtaining an approximation algorithm from a type A relaxed decision procedure was given for machine scheduling by Lenstra, Shmoys and Tardos [23].

Theorem 1.6. *Let $\mathcal{O}$ be a searchable optimization problem and suppose that $\mathcal{O}$ has a polytime type A p-relaxed decision procedure $\mathcal{A}$. Then $\mathcal{O}$ has a p-approximation algorithm.*

Proof. We assume that $\mathcal{O}$ is a maximization problem satisfying the conditions. Let $\mathbf{P}$ be an instance of $\mathcal{O}$ with input size n, so we have an integer polynomial $q(n)$ bounding the number of bits in the binary representation of the optimal value for $\mathbf{P}$.

Let $f(x,y) := \lceil \frac{x+y}{2} \rceil$ be the function which takes the ceiling of the average of a pair of integers. Consider the following algorithm $\mathcal{A}'$ based on $\mathcal{A}$, which runs as follows on input a problem instance $\mathbf{P}$ for $\mathcal{O}$ with size n:
1. Initialize lower:= 0 and upper:= $2^{q(n)+1}$.
2. While lower≠upper
   3. Run $\mathcal{A}$ on ($\mathbf{P}$, f(lower,upper)).
   4. If $\mathcal{A}$ returns "No" update upper:= f(lower,upper)-1.
   5. Else update lower:= f(lower,upper).
6. End While.
7. Run $\mathcal{A}$ on ($\mathbf{P}$, lower).
8. If $\mathcal{A}$ returned "Almost" and a solution x for $\mathbf{P}$ then **return** x.
9. Else **return** "Error".

$\mathcal{A}'$ runs a binary search for the maximum integer T for which algorithm $\mathcal{A}$ returns "Almost" and outputs the corresponding feasible solution for $\mathbf{P}$ with objective value at least $\frac{T}{p}$. Indeed the lower bound is initialized at 0, which is by non-negativity of objectives below the maximum T, and is adjusted up to values $\mathcal{A}$ is known not to give response "No" to. On the other hand the upper bound is set initially to a bound on the best possible objective value, and is only decreased to one below T for which $\mathcal{A}$ is known to return

"No". Therefore as claimed $\mathcal{A}'$ finds this maximum T and gives a feasible solution with objective at least $\frac{T}{p}$. From optimal objective integer and Definition 1.4 ensuring that a "No" response indicates infeasibility for the target, this implies that this maximum T is at least the optimal value for $\mathbf{P}$, so the obtained feasible solution is within a factor of p of optimal.

Since the upper bound for the binary search had a polynomially bounded number of bits, the search runs in a polynomially bounded number of steps, thus running in polytime (since the steps have polytime runtime by assumption on $\mathcal{A}$). Therefore $\mathcal{A}'$ is a p-approximation algorithm for $\mathcal{O}$ as desired. □

We now introduce the analogous result for the type B relaxed decision procedure.

Theorem 1.7. *Let $\mathcal{O}$ be a searchable optimization problem and suppose that $\mathcal{O}$ has a polytime type B p-relaxed decision procedure $\mathcal{B}$. Then $\mathcal{O}$ has a polytime algorithm which takes problem instances $\mathbf{P}$ of $\mathcal{O}$ as input and finds a value b such that the optimal value for $\mathbf{P}$ is guaranteed to lie in the interval $[b, pb]$.*

Proof. We can assume $\mathcal{O}$ is a maximization problem and proceed exactly as in the previous proof, by considering problem instance $\mathbf{P}$ with optimal value an integer of bitlength at most $q(n)$.

Let $f(x, y) := \lceil \frac{x+y}{2} \rceil$ be as before. Consider the following algorithm $\mathcal{B}'$ based on $\mathcal{B}$, which runs as follows on input a problem instance $\mathbf{P}$ for $\mathcal{O}$ with size n:

1. Initialize lower:= 0 and upper:= $2^{q(n)+1}$.
2. While lower≠upper
   3. Run $\mathcal{B}$ on ($\mathbf{P}$, f(lower,upper)).
   4. If $\mathcal{B}$ returns "No" update upper:= f(lower,upper)-1.
   5. Else update lower:= f(lower,upper).
6. End While.
7. **Return** $\frac{1}{p}$(upper).

$\mathcal{B}'$ runs a binary search for the maximum integer T for which algorithm $\mathcal{B}$ returns "Almost" and outputs value $\frac{T}{p}$. The justification for this is the same as in the previous result. As before this maximum T bounds the optimal value for $\mathbf{P}$. Since $\mathcal{B}$ running on ($\mathbf{P}$, T) gives "Almost", by Definition 1.4 this guarantees us that there exists a feasible solution to $\mathbf{P}$ with objective at least $\frac{T}{p}$. Therefore the optimal value for $\mathbf{P}$ lies in the range $[\frac{T}{p}, T]$ the bounds of which are within a factor of p of each other.

Also as in the prior result, the runtime for this is polytime and therefore the algorithm $\mathcal{B}'$ is as desired. □

We say that an optimization problem has an optimal value that can be *estimated* within a factor of p in polytime, when an interval of form $[b, pb]$ can be constructed in polytime, within which the optimal value is known to reside.

The difference between the two types of p-relaxed decision procedures, is that the A-type provides a way to get from a problem instance with a feasible target to a solution

within p of the target, while the B-type accurately reports that there exists a feasible solution within p of the target. This corresponds to the difference in being able to provide an algorithm for getting approximately optimal solutions and merely being able to estimate the optimum.

By restricting ourselves to rational input for (non-infinite) v_{ij}/p_{ij} values, and scaling, we may insist upon integer values for the machine scheduling and Santa Claus problems. This will result in integer objective values of polynomially bounded bitlength for all feasible solutions to any problem instance (in particular $\sum_{i\in P, j\in R} v_{ij}$ and $\sum_{i\in M, j\in J: p_{ij}\neq\infty} p_{ij}$ are trivial bounds with polynomial bitlength). Thus integer valued machine scheduling and Santa Claus problems are *searchable optimization problems*. Therefore the preceding Theorems 1.6 and 1.7 apply to our problems.

These facts have been well understood by those involved in the analysis of approximation for optimization problems, but in writing this thesis it was found useful to introduce this terminology and these basic results. In particular it is helpful to unify the techniques involved in, and to sharpen the distinction between, being able to get approximate solutions and being able to get estimates for the optimal value. These relaxed decision procedures will be the method by which results for approximation and estimation are obtained for these problems. While the results concerning them are basic applications of binary search, it is hoped that it will be worthwhile and beneficial to understanding to have outlined these two notions of relaxed decision procedures explicitly, and in a general form applicable to a wide variety of optimization problems.

Chapter 2

Early Work on Machine Scheduling

We now discuss a 1990 paper by Lenstra, Shmoys and Tardos [23] which concerns the machine scheduling problem with integer valued performance times and provides many of the best known results to date. By scaling, we assume all job performance times to be integer (or infinite). Having integer performance times assures us the optimal makespan is integer and makes the machine scheduling problem a *searchable optimization problem* (see Definition 1.2).

The most notable positive result from the paper is a 2-approximation algorithm produced for the machine scheduling problem, which was the first to establish that the problem is approximable within a constant factor. Rather remarkably, in general this still gives the best factor of approximation among any known algorithm to date. Accompanying this is a $\frac{3}{2}$ factor NP-hardness result for estimating the optimal value of the machine scheduling problem within a factor of $\frac{3}{2}$ and this hardness result holds even in the restricted assignment case. This leaves a, still standing, $\frac{1}{2}$ difference in the gap between the factors 2 and $\frac{3}{2}$ in which the problem is neither known to be polytime approximable or inapproximable.

The approach of the paper was to produce a type A 2-relaxed decision procedure (in this paper [23] it is simply called a 2-relaxed decision procedure) for the machine scheduling problem and then justify that this is enough to yield the corresponding approximation result. We have already developed a general version of a statement to that effect (see Theorem 1.6), so we proceed to examine how the type A 2-relaxed decision procedure is constructed.

We consider the following pair of linear programs defined for a problem instance $\boldsymbol{P}$, and two parameters we introduce as the target deadline $d \in \mathbb{Z}^+$ and the time threshold $t \in \mathbb{Z}^+$ for a single job. We will seek to round feasible solutions of the one linear program to integer solutions to the other.

Given machine $i \in M$ and time threshold t, we define $J_i(t)$ to be the set of jobs performable on machine i within the threshold t. That is $J_i(t)$ consists of those jobs $j \in J$ with $p_{ij} \leq t$. Similarly given time t and job j we define $M_j(t)$ to be the set of machines capable of performing job j within the threshold t, those machines $i \in M$ with $p_{ij} \leq t$.

LP($\boldsymbol{P}$,d,t)

$$\sum_{i \in M_j(t)} x_{ij} = 1 \qquad \forall j \in J \tag{2.1}$$

$$\sum_{j \in J_i(t)} p_{ij} x_{ij} \leq d \qquad \forall i \in M \tag{2.2}$$

$$\mathbb{R} \ni x_{ij} \geq 0 \qquad \forall i \in M, j \in J_i(t). \tag{2.3}$$

The above linear program (2.1)-(2.3) is a relaxation of the task of creating a job assignment that has all machines meeting the deadline d, while only processing jobs on machines that will perform them in time at most t. It has variables x_{ij} for each pair $i \in M$ and $j \in J$ so that job j is performable on machine i within time t, which are representing weights of a fractional job assignment. The constraints insist that each job is given total weight 1 in the assignment and that the weighted (by x_{ij}) sum of performance times for each machine is at most the deadline time d. So $x_{ij} \in \{0, 1\}$ solutions correspond to job assignments meeting the target deadline, by viewing $x_{ij} = 1$ as an assignment of job j to machine i. Note that finding a feasible solution, or certifying infeasibility, to this linear program is achievable in polytime since this is a linear program with number of variables and constraints each polynomial in $|M|$ and $|J|$.

IP($\boldsymbol{P}$,d,t)

$$\sum_{i \in M_j(t)} x_{ij} = 1 \qquad \forall j \in J \tag{2.4}$$

$$\sum_{j \in J_i(t)} p_{ij} x_{ij} \leq d + t \qquad \forall i \in M \tag{2.5}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall i \in M, j \in J_i(d). \tag{2.6}$$

This integer program (2.4)-(2.6) corresponds to the task of creating a job assignment that has all machines meeting deadline $d + t$ with the restriction that all jobs must be assigned to a machine capable of performing it in time t, by viewing $x_{ij} = 1$ as assignment of job j to machine i. The first set of constraints captures the requirement that every job gets assigned, and the second set of constraints ensures that the set of jobs assigned to each machine has total time at most the deadline $d + t$.

Proposition 2.1. *[23] Given problem instance $\boldsymbol{P}$, deadline $d \in \mathbb{Z}^+$ and time threshold t so that LP($\boldsymbol{P}$,d,t) is feasible, we can obtain a feasible solution to integer program IP($\boldsymbol{P}$,d,t) in polytime.*

At this juncture insisting on integer performance times is needed for this rounding. We omit the proof of this result, but shall briefly sketch some aspects of the argument. An extreme point x of LP($\boldsymbol{P}$,d,t) is found in polytime and an associated bipartite graph

$(M \cup J, E)$ is considered, which has edges between machine i and job j if and only if j is assigned with fractional weight to i. By exploiting properties of extreme points it is concluded that the resulting graph has components which are either trees, or have exactly one cycle (a tree plus an edge). By rooting the tree components and assigning each job to a machine that is one of its children, we can create an assignment of jobs that at most assigns the additional load of one job to each machine. Job vertices will not be leaves, since to appear in this graph by definition it must be fractionally assigned to (hence adjacent to) multiple machines. So a choice of a child machine is always available for the assignment described above.

The single cycle components are also reduced to this case by assigning jobs in the cycle according to an arbitrary choice of orientation for the cycle. The components created by deleting cycle edges may then be rooted at each of the cycle vertices and we can assign jobs in each resulting tree by choosing a child machine as before. We note that t bounds performance times among the options for fractional assignment used in LP($\mathbf{P}$,d, t). Our described rounding to a pure assignment gives at most one of these fractionally assigned (by x) jobs to a given machine, hence adding load at most t on any machine. So a pure integer assignment of jobs is created that has makespan at most $d + t$, thus creating a feasible solution to IP($\boldsymbol{P}$,d,t) in polytime.

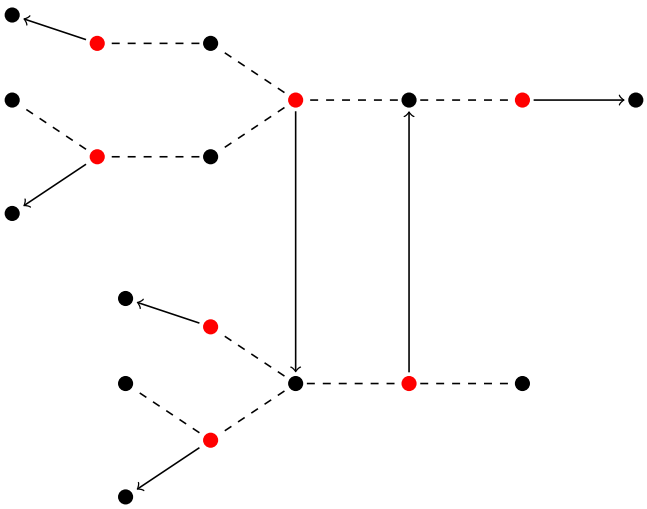

Figure 2.1: (Proposition 2.1) The rounding to a solution of IP($\mathbf{P}$,d, t) that occurs in a cycle component of the bipartite graph constructed from an extreme point of LP($\mathbf{P}$,d, t). In black are machine vertices, in red job vertices, with arcs indicating assignment of jobs to machines, and dashed edges indicating the remaining edges of the component.

Somewhat similar ideas (see work by Ebenlendr, Krčál and Sgall [12]) of assigning jobs by rooting a tree of fractional assignments will be seen in the *graph balancing* case of the machine scheduling problem where jobs are assignable to at most 2 machines and the problem is naturally interpretable as orienting the edges, which correspond to jobs, into vertices for the machines.

Given this we can now describe an algorithm given input problem instance $\boldsymbol{P}$ and deadline d, which outputs as follows: If LP($\boldsymbol{P}$,d,d) is infeasible output "No." Otherwise we

proceed as in Proposition 2.1, by finding an extreme point solution to LP($\boldsymbol{P}$,d,d) and we round it in polytime to a feasible solution to IP($\boldsymbol{P}$,d,d). The corresponding job assignment to this feasible solution is returned as output along with a response "Almost".

Lemma 2.2. *[23] The above algorithm is a polytime type A 2-relaxed decision procedure for the machine scheduling problem.*

Proof. LP($\boldsymbol{P}$,d,d) can have feasibility tested in polytime and by Proposition 2.1, the rounding to a feasible solution of IP($\boldsymbol{P}$,d,d) (with direct correspondence to a job assignment) may also be achieved in polytime. Therefore the algorithm runs in polytime.

To meet deadline d, no job may be assigned to a machine which performs it in time greater than d. Therefore LP($\boldsymbol{P}$,d,d) is a relaxation of the task of finding a job assignment meeting deadline d, and its infeasibility tells us that there is no job assignment with makespan at most d. So a "No" output indeed certifies infeasibility of finding a job assignment. Otherwise the job assignment produced by the algorithm corresponds to a feasible solution to IP($\boldsymbol{P}$,d,d) and is therefore a job assignment with makespan at most $2d$. □

Theorem 2.3. *[23] There is a 2-approximation algorithm for the machine scheduling problem.*

Proof. This is an immediate consequence of the preceding Lemma and Theorem 1.6 for relaxed decision procedures. □

Let $m = \max_{i \in M, j \in J} p_{ij}$. Note that we can use the rounding from Proposition 2.1 to obtain a type A $(1+m)$-relaxed decision procedure for machine scheduling as well [23], and hence a $1+m$ approximation algorithm. In general this is worse, as some job performance times on given machines may be greater than the optimal makespan, but for the restricted case with job sizes fixed it yields the same result. In cases where job sizes are known to be small relative to the optimal makespan this yields superior results, and demonstrates (given the $\frac{3}{2}$ hardness result discussed below) that the difficulty of the approximation problem comes from the large jobs.

Complementing the approximation algorithm above was a $\frac{3}{2}$ approximation hardness result that applies even for the restricted case.

Theorem 2.4. *Estimating the optimal makespan for the restricted machine scheduling problem strictly within a factor of $\frac{3}{2}$ is NP-hard.*

The result shows that not only is a $\frac{3}{2} - \epsilon$ approximation algorithm in polytime not possible for any $\epsilon > 0$ (given $P \neq NP$), but it is not even possible to in polytime produce bounds strictly within a $\frac{3}{2}$ factor in which the optimal makespan can be guaranteed to lie. The hardness result is based on the well-known NP-completeness of the 3-dimensional perfect matching problem [23] which we describe below.

Definition 2.5. Suppose we are given a *triple system* consisting of disjoint sets A, B, C of size n and set of triples $\mathcal{T}$ with one entry in each set. The *3-d matching problem* is the task of determining whether $\mathcal{T}$ contains a disjoint set of n triples whose union is all of $A \cup B \cup C$ (such a set of triples is called called a *perfect matching* for the system, see Figure 2.2 for an example).

This problem is well known to be NP-complete, being one of Karp's 21 NP-complete problems [21].

To obtain a reduction from this 3-d matching problem, we proceed from an arbitrary instance of this 3-d matching problem and construct a restricted assignment machine scheduling problem which has an assignment with makespan 2 when there is a perfect matching in the triple system, and otherwise has makespan at least 3. Thus to determine the optimal makespan strictly within a factor of $\frac{3}{2}$ amounts to solving the 3-d matching problem, giving the desired hardness result. This remains the best hardness result obtained for even the general case of polynomial time approximations to the optimal makespan.

Similarly for the restricted case of the Santa Claus Problem we can also get a hardness result for estimating the optimal value strictly within a factor of 2 (as given by Bezáková and Dani [6]) by reduction to 3-d matching. We exhibit the reduction for the Santa Claus problem as follows, which uses the same ideas [23] as for machine scheduling.

Defining a correspondence to Santa Claus problem: Suppose we are given an instance of the 3-d matching problem with sets A, B, C and triples $\mathcal{T}$, with number of triples hitting each $a \in A$ given by n_a (we assume $n_a \geq 1$ as determining there is no perfect matching is trivial in that case). We proceed to construct a corresponding instance of the restricted Santa Claus problem. Players are identified with triples by $P := \mathcal{T}$. The set of resources R consists of the elements of $B \cup C$ together with $n_a - 1$ resources identified with each $a \in A$:

$$R := B \cup C \cup \{a_k : a \in A \text{ and } k \in \{1, \ldots, n_a - 1\}\}.$$

The resources a_k identified with each $a \in A$ have value 2 and are restricted to be assigned to those players/triples which contain a. The other resources each have value 1 and are restricted to assignment to exactly those players/triples containing it.

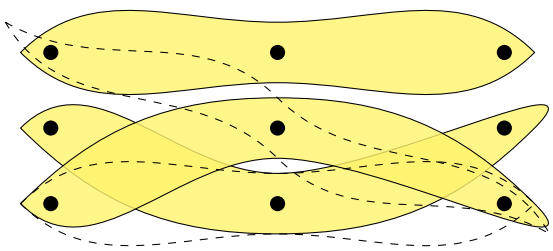

Figure 2.2: A perfect matching for a triple system is illustrated with columns corresponding to sets A,B and C, with chosen triples in yellow. Unused triples of $\mathcal{T}$ are marked by dashed borders.

Lemma 2.6. *$\mathcal{T}$ contains a perfect matching for A, B, C exactly when we have an allocation of resources that allocates at least value 2 to each player in the corresponding Santa Claus problem instance.*

Proof. If $\mathcal{T}$ contains a perfect 3-d matching $M \subset \mathcal{T}$ then we may assign all resources j in B or C to the player given by the unique triple of M containing j. This allocates value 2 to all of the players in M, and for every player in $\mathcal{T} \setminus M$ containing $a \in A$ we can assign one of the resources a_k identified with a giving value 2. Since M hits every $a \in A$ uniquely, we have that $T \setminus M$ has $n_a - 1$ triples containing a, so we have enough copies of resources identified with a so that all the copies can be assigned uniquely. Thus we have an allocation of resources giving at least 2 value to all players.

Suppose now that we have an allocation of resources giving value at least 2 to all players. For each $a \in A$ we have only $n_a - 1$ copies of resources identified with a. So there exists some player/triple containing a which is not allocated a resource of form a_k and we pick one, say p_a. Player p_a only values the resources from B and C it contains, in addition to those of form a_k (none of which it is allocated). Since p_a is given value at least 2, it must then be allocated both of the resources from B and C that it contains. Resources are assigned to at most one player, therefore $\{p_a : a \in A\}$ is a disjoint set of triples and hence a perfect matching for the triple system. □

Furthermore, because all resources have integer value this Lemma says that $\mathcal{T}$ does not contain a perfect matching exactly when any allocation must allocate 1 or less to some player. So estimating (strictly) within a factor of 2 the max-min of this instance of the Santa Claus Problem tells us whether our original instance of a 3-d matching problem admits a perfect matching. Hence from the NP-completeness of determining whether or not we have a perfect 3-d matching, we obtain the following hardness result for the restricted case of the Santa Claus problem.

Theorem 2.7. *Estimating the max-min for the restricted case of the Santa Claus problem strictly within a factor of 2 is NP-hard.* □

Thus given $P \neq NP$ we have that we cannot in polytime estimate the max-min for the restricted case of the Santa Claus problem within a factor of 2. So we obtain an estimation hardness result for the Santa Claus problem using the same techniques as used for machine scheduling.

Bezáková and Dani considered [6] an approximation algorithm for the Santa Claus problem based on the techniques in the 2-approximation algorithm for machine scheduling seen here. Let $v_{max} = \max_{i \in P, j \in R} v_{ij}$ be the max valuation given to any resource by a player and let OPT be the (optimal value) max-min for the problem instance. A polytime approximation algorithm [6] is given that produces an allocation with min value bundle at least value $OPT - v_{max}$. In general this guarantee says nothing, as even in the restricted case $OPT - v_{max}$ may be 0 or negative. So we examine a different linear program relaxation to study the Santa Claus problem in the following chapter.

Chapter 3

Santa Claus CLP

The corresponding linear program relaxations that were used for machine scheduling behave poorly for the Santa Claus problem when there are single resources that are valued close to the max-min. So a new LP relaxation called the Configuration LP (CLP) was introduced by Bansal and Sviridenko [5] whose integral solutions (for CLP with parameter T) correspond to allocations that give value at least T to all players. In the restricted case, where all resources/jobs have inherent sizes, the CLP has been useful in obtaining a constant factor of estimation for the max-min [4, 18].

The results discussed in this section for the Santa Claus CLP will also be applicable, with trivial alteration, to the machine scheduling version as well. The corresponding machine scheduling CLP has also been used [28, 19] in the restricted case to improve upon the estimation factor of 2 for the optimal makespan that comes from the 2-approximation algorithm previously discussed [23].

Definition 3.1. For $i \in P$ let $C(i,T) := \{S \subset R : \sum_{j \in S} v_{ij} \geq T\}$ be the set of bundles of resources with total value to player i at least T. We call such bundles *configurations* for player i and target T. For the restricted version, we omit resources not assignable to each player i (0-valued) from its corresponding configurations.

The CLP and its dual are then given as follows for an instance of the Santa Claus problem and parameter $T > 0$ representing our target allocation.

Santa Claus CLP

$$\min \quad 0 \tag{3.1}$$

$$\sum_{S \in C(i,T)} x_{i,S} \geq 1 \qquad \forall i \in P \tag{3.2}$$

$$\sum_{i,S: j \in S, S \in C(i,T)} x_{i,S} \leq 1 \qquad \forall j \in R \tag{3.3}$$

$$x_{i,S} \geq 0 \qquad \forall i \in P, S \in C(i,T). \tag{3.4}$$

The CLP has non-negative variables, for every pair (i,S) with S a bundle satisfying target T for player i, representing weights of a fractional allocation of bundles. It has

constraints requiring that every player be given a total weighting at least 1 on the bundles it is allocated and is constrained to have no resource allocated with total weight exceeding 1. That is we are required to not under-serve any player, nor to over-assign any resource. We also give it a trivial linear minimization objective of 0, so this is a linear program with a corresponding dual.

Santa Claus Dual CLP

$$\max \quad \sum_{i \in P} y_i - \sum_{j \in R} z_j \tag{3.5}$$

$$\sum_{j \in S} z_j \geq y_i \qquad \forall i \in P, S \in C(i,T) \tag{3.6}$$

$$y_i, z_j \geq 0 \qquad \forall i \in P, \forall j \in R. \tag{3.7}$$

The dual has non-negative variables for each player and for every resource, together with constraints for every pair (i, S) with S a bundle satisfying target T for player i. Note that by *weak duality*, a feasible dual solution with positive objective certifies the infeasibility of the primal CLP. The constraints of the dual are insensitive to a scaling of (y, z) to $\lambda(y, z)$, while the objective scales with λ. Therefore a positive objective solution for the dual also implies the unboundedness of the dual program. Also taking all variables 0 is feasible with objective 0. So by *strong duality* the primal CLP is feasible if and only if the dual has optimal objective value 0.

An allocation of resources $\{S_i\}_{i \in P}$ satisfying target T corresponds to an integral solution of the CLP x where $x_{i,S} = 1$ if and only if $S = S_i$ and $x_{i,S} = 0$ otherwise. To obtain an allocation distributing at least T to all players from a feasible integral solution to the CLP for target T we do the following: for each player i we arbitrarily pick one bundle S such that $x_{i,S}$ is non-zero in the feasible solution, and allocate this S to player i. From the constraints on weighting for each player, there is at least one such bundle to pick for each player and from the constraints on weighting for each resource, the resulting bundles in our allocation are also disjoint. So in this way we obtain an allocation of satisfying (for target T) bundles for all players. So the CLP (3.1)-(3.4) is indeed a relaxation the Santa Claus problem.

Note that in relation to the size of the information which specifies the problem (the valuations of the resources by the players) there are potentially exponentially many variables in the CLP, however the program and its dual have nice properties that can be exploited [5] to make it more tractable.

Definition 3.2. [30] A *separation oracle* for a linear program (LP) is an algorithm which given a candidate solution either certifies its feasibility for (LP), or returns a violated constraint of (LP).

Using the ellipsoid method, a linear program with a polynomially bounded number of variables and a polytime separation oracle may be solved in polytime (this result is a consequence of work by Khachiyan in 1979 on the ellipsoid method applied to linear programming [22]). We can also assume input to the separation oracle (as called by the ellipsoid method) is rational.

Definition 3.3. The *regular minimum knapsack problem* is the following linear program optimization problem with positive integer data k, M and $(a_j, c_j)_{j\in\{1,\dots,k\}}$

Regular Min Knapsack Problem

$$\min \sum_{j\in\{1,\dots,k\}} c_j x_j \tag{3.8}$$

$$\sum_{j\in\{1,\dots,k\}} a_j x_j \geq M \tag{3.9}$$

$$x_j \in \{0,1\} \qquad \forall j \in \{1,\dots,k\}. \tag{3.10}$$

While this knapsack problem is NP-Complete [21], it is known to be solvable in *pseudo-polytime* [30], meaning in particular that it is solvable in polytime with respect to nM, where n is the input size of the integer data, by applying dynamic programming techniques [30].

Proposition 3.4. *[5] The Santa Claus Dual CLP (3.5)-(3.7) with integer valuations and integer target T has a polytime in nT separation oracle, where n is the input size specifying the problem instance and the candidate solution given to the oracle.*

Proof. Fix an instance of the Santa Claus problem with integer valuations and an integer target T. Let (y, z) be a candidate solution to (3.5)-(3.7) with rational entries given in binary. Scaling does not affect feasibility of a candidate solutions to the dual, so we may assume that (y, z) is integer valued.

We label our resources as $R = \{1, \dots, k\}$ and for each player $i \in P$ we consider a corresponding regular min knapsack problem on data using the same k, $M := T$, $a_j := v_{ij}$ and $c_j := z_j$. Note that S is a configuration for player i if and only if its indicator function $x = (x_j)_{j\in R}$ satisfies $\sum_{j\in R} v_{ij}x_j \geq T$ the constraint for feasibility in the corresponding regular min knapsack problem. Furthermore, a configuration S for i has its corresponding dual constraint (3.6) violated if and only if its indicator function x satisfies $\sum_{j\in R} z_j x_j < y_i$. Therefore (i, S) corresponds to a violated dual constraint if and only if its indicator function is a feasible solution to the corresponding regular min knapsack problem with objective value less than y_i. For each player $i \in P$ we solve the corresponding regular min knapsack problem, determining optimal value w_i and optimal solution x^i with corresponding set of resources $S^i := \{j \in \{1, \dots, k\} : x^i_j = 1\}$. Therefore determining that $w_i \geq y_i$ for all $i \in P$ certifies feasibility of (y, z) for the dual CLP on target T, and finding $w_i < y_i$ gives us a violated dual constraint corresponding to (i, S^i) that we return. Note that the input data for each knapsack problem considered is contained in the given data to the separation oracle and in each case the target M agrees with T. Thus by solving these knapsack problems corresponding to each player, we have produced a polytime separation oracle in nT as desired. □

As sketched in the paper introducing the Santa Claus CLP [5], the previous result gives a way to solve the primal problem with the same runtime guarantee.

Theorem 3.5. *[5] The Santa Claus CLP (3.1)-(3.4) for any problem instance with integer valuations and integer target T is solvable in polytime with respect to T and the input size n, producing either a feasible solution or a verification of infeasibility.*

Proof. Using the ellipsoid method and the described separation oracle for the dual CLP (3.5)-(3.7), we can in polytime (in T and n) solve the dual program after consideration of a polynomially bounded number of inequality constraints (since the ellipsoid method runs in polytime) labelled by $U \subset \bigcup_{i \in P}\{(i,S) : S \in C(i,t)\}$.

Either we conclude that the dual CLP is unbounded, thus verifying infeasibility for the primal CLP, or we have produced an optimal solution (y^*, z^*). As discussed earlier this must have objective value 0 and certify the feasibility of the primal. Moreover, as U consists of the constraints that were needed to verify optimality for (y^*, z^*) in the ellipsoid method, the primal must have a feasible solution x such that $x_{i,S} = 0$ for all $(i,S) \notin U$. This holds by applying strong duality to the primal-dual pair consisting of this CLP keeping only U variables and the dual to the CLP with only constraints from U. So we can solve the primal CLP by setting all variables not indexed by U to 0, and then solve the resulting linear program with a polynomially bounded number of variables and constraints. Thus in polytime in n and T we have solved the Santa Claus CLP. □

The integrality gap of the CLP (3.1)-(3.4) refers to the factor between the optimal sizes of T for which the CLP admits feasible real solutions and feasible integral solutions. Given the correspondence discussed above between integral solutions and satisfying allocations, this is also the gap between the optimal T for which the CLP relaxation is feasible and the max-min for the Santa Claus problem.

Theorem 3.6. *Suppose a class of instances of the Santa Claus problem have corresponding CLPs (3.1)-(3.4) with integrality gap at most p, then there exists a type B $(p+\epsilon)$-relaxed decision procedure for this class of instances for any $\epsilon > 0$.*

Proof. Let $\epsilon > 0$ and we choose $\delta = \frac{\epsilon}{p+\epsilon}$, so that $\frac{1}{p+\epsilon} = \frac{1-\delta}{p}$. Suppose we are given problem **P** from this class of instances of the Santa Claus problem and a target $T > 0$ as input, we describe an algorithm $\mathcal{B}$ which produces "No" or "Almost" as follows.

The idea of the algorithm will be to scale our problem and round the target allocation T to some integer of polynomially bounded size, so that we can apply the previous results to determine feasibility of the corresponding CLP for the new instance **P'** in polytime. By relating the feasibility of **P** and **P'** on different targets, and using the known bound on integrality gap for **P**, this will allow the polytime test of feasibility to suffice as a type B relaxed decision procedure.

Let k be the number of resources in **P** and let $f = \frac{\delta T}{k}$. We consider Santa Claus instance **P'** determined from **P** with same players and resources, with new valuations $v'_{ij} := \lceil \frac{v_{ij}}{f} \rceil$ for all $i \in P$ and $j \in R$ scaled by $\frac{1}{f}$ and rounded up.

Let $T' = \lceil \frac{T}{f} \rceil$ be the ceiling of $\frac{T}{f} = \frac{k}{\delta}$. All valuations for **P'** are integer and T' is polynomial in input size, therefore by Theorem 3.5 we may determine whether the CLP

(3.1)-(3.4) for **P'** with target T' is feasible in polytime. If it is not feasible $\mathcal{B}$ outputs "No", otherwise $\mathcal{B}$ outputs "Almost".

Note that algorithm $\mathcal{B}$ runs in polytime. Suppose $\mathcal{B}$ outputs "No" and hence that the CLP for **P'** with target T' is infeasible. Firstly note that from the valuations being integer, the CLP for **P'** being infeasible with target T' implies it is infeasible for target $\frac{T}{f}$. Since values were scaled and rounded up, we have that a bundle of resources being satisfying for a player with target T in instance **P** implies the same bundle of resources satisfies that player with target $\frac{T}{f}$ in instance **P'**. Thus infeasibility of the CLP for **P'** with target $\frac{T}{f}$ implies infeasibility of the CLP relaxation for **P** with target T. Therefore $\mathcal{B}$ outputting "No" implies that no solution to the Santa Claus problem **P** has objective value at least T.

Now suppose that $\mathcal{B}$ outputs "Almost". This implies that **P'** has its CLP feasible for target $T' \geq \frac{T}{f}$. Let v_{ij} and v'_{ij} be the valuations of resource j by player i for **P** and **P'** respectively. We note that $v'_{ij} < 1 + \frac{v_{ij}}{f}$ and therefore $v_{ij} > fv'_{ij} - f$. So a bundle of (at most k) resources satisfying target $\frac{T}{f}$ for some player in instance **P'**, implies the same bundle is satisfying for that player with target $T - kf$ for **P**. Therefore **P** is feasible for $T - kf = T(1 - \delta)$. Since the integrality gap for **P** is at most p, this shows that **P** has an allocation satisfying target $\frac{T(1-\delta)}{p} = \frac{T}{p+\epsilon}$.

Therefore $\mathcal{B}$ is a type B $(p + \epsilon)$-relaxed decision procedure for this class of instances as desired. □

So by Theorem 1.7, this shows that we can estimate the max-min arbitrarily close to within the integrality gap factor for the CLP. This does not suffice for finding approximately optimal solutions, as it may be difficult to round solutions to the CLP with target T to integral solutions for a target within the same factor.

Theorem 3.7. *Let $\mathcal{O}$ be a class of instances of the Santa Claus problem such that there is a polytime algorithm $\mathcal{A}$, which given a problem instance and a solution to the CLP (3.1)-(3.4) for target T, finds an allocation of resources meeting target $\frac{T}{p}$ for all players. Then there exists a type A $(p + \epsilon)$-relaxed decision procedure for $\mathcal{O}$ and any $\epsilon > 0$.*

We omit the proof, it follows in essentially the same way as the prior result, where we pair the "Almost" with an allocation determined by $\mathcal{A}$. Again, combining with our discussion of relaxed decision procedures (see Theorem 1.6) this yields a $p+\epsilon$ approximation for any such cases of the Santa Claus problem.

The proof of Theorem 3.6 may also be modified to show that the optimal T for which the CLP is feasible may be determined to arbitrary precision in polytime, as argued originally by Bansal and Sviridenko [5]. This fact can also be seen to imply the previous results on max-min estimation and approximation.

The CLP can be equivalently formulated using equality constraints as follows.

Santa Claus Equality CLP

$$\min 0 \tag{3.11}$$

$$\sum_{S \in C(i,T)} x_{i,S} = 1 \qquad \forall i \in P \tag{3.12}$$

$$\sum_{i,S: j \in S, S \in C(i,T)} x_{i,S} = 1 \qquad \forall j \in R \tag{3.13}$$

$$x_{i,S} \geq 0 \qquad \forall i \in P, S \in C(i,T). \tag{3.14}$$

Note that this is also a relaxation of the Santa Claus problem, with feasible integral solutions existing if and only if there exists an allocation satisfying target T. The correspondence of an integral solution to a satisfying allocation is exactly as in the inequality version. Also, given a satisfying allocation for target T, we can assume that all resources are put in some bundle of the allocation (by assigning unused resources arbitrarily to players), thus giving equality for resource constraints when setting $x_{i,S}$ 1 or 0 based on whether S is the bundle allocated to player i. Since bundles are purely allocated to players with weight 1 this also satisfies the equality constraints for players. The corresponding dual program is as follows:

Dual Santa Claus Equality CLP

$$\max \quad \sum_{i \in P} y_i - \sum_{j \in R} z_j \tag{3.15}$$

$$\sum_{j \in S} z_j \geq y_i \qquad \forall i \in P, S \in C(i,T) \tag{3.16}$$

$$y_i, z_j \text{ free in } \mathbb{R} \qquad \forall i \in P, \forall j \in R. \tag{3.17}$$

Equality versions of the CLP will be used in our later Chapter 7 on graph balancing versions for the Santa Claus and machine scheduling problems. The results discussed for the inequality version (3.1)-(3.4) apply here as well, where the corresponding knapsack problem for the dual separation oracle is allowed negative integer data.

The CLP and its dual have proved fruitful in the analysis of the Santa Claus problem, especially in the restricted case. Bansal and Sviridenko showed [5] for the general case, that the CLP still has unbounded integrality gap, so for constant factor estimation/approximation it is generally unsuitable (and indeed the general Santa Claus problem has not been shown to even have a constant factor estimation algorithm). In the restricted case it was shown, in a paper due to Asadpour, Feige and Saberi [4], that the Santa Claus CLP always has an integrality gap of at most 4, implying that the max-min to any restricted Santa Claus problem may be estimated in polytime to within a factor arbitrarily close to 4. The proof [4] draws from alternating path-like techniques seen in a proof [15] of Haxell's Theorem, which is a generalized version of Hall's Theorem applicable to bipartite hypergraphs.

In Chapter 4 we shall give a novel proof which obtains this result from the Santa Claus CLP using a version of Haxell's Theorem directly to produce a positive solution to the dual in order to certify infeasibility of the primal. Constructing dual solutions with objective value to certify infeasibility of the primal is a technique, which was also applied (by Svensson [28]) to the analogous linear program relaxations of the restricted machine scheduling problem to obtain improved estimation results, as we shall later discuss. This factor of 4 for estimation and integrality gap of the Santa Claus CLP has also been more recently improved to $4-\frac{1}{6}$ in work due to Jansen and Rohwedder [18], and then to $(4-\frac{5}{26})$ by Cheng and Mao in 2019 [9].

Chapter 4

Bipartite Hypergraphs and Independent Transversals

4.1 A Bipartite Hypergraph Framework

We introduce the notion of bipartite hypergraphs here as they provide another natural way to formulate our two optimization problems. We will also discuss a result (Haxell's Theorem [15]) pertaining to bipartite hypergraphs, which was obtained using techniques similar to some of those later applied to the study of these two optimization problems. A version of Haxell's Theorem is also applied directly in this section to obtain an estimation result for the restricted Santa Claus problem in a novel way.

Definition 4.1. A *hypergraph* is a pair $\mathcal{H} = (V, \mathcal{E})$ of vertices V, together with a set of (hyper)edges $\mathcal{E}$ each of which is a subset of V.

Just as in graphs, a *matching* in a hypergraph is a set of vertex disjoint edges. A set of edges $\mathcal{U}$ is said to *cover* or *saturate* any set of vertices which is contained in the union over the set of edges in $\mathcal{U}$.

Definition 4.2. A *transversal* of a hypergraph is a set of vertices that intersects every edge.

Definition 4.3. [4] Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph. When $V = A \cup X$ is a disjoint union of A and X such that every edge has exactly one vertex from A, then the hypergraph $\mathcal{H}$ is said to be bipartite with bipartition (A, X). We also write $\mathcal{H} = (A \cup X, \mathcal{E})$ to denote a *bipartite hypergraph* with bipartition (A, X).

To see how bipartite hypergraphs relate to our two problems, we consider the decisional variant of trying to allocate bundles of items that meet a given target requirement T, for either a target makespan we should not exceed in any bundle, or a target value for allocation we must at least meet in bundles given to each player.

Recall Definition 3.1 for $C(i,T)$ on an instance of the Santa Claus problem for player i and target T. We analogously define $C(i,T)$ for machine scheduling by setting:

$$C(i,T) := \{S \subset J : \sum_{j \in S} p_{ij} \leq T\} \tag{4.1}$$

to be the set of all bundles of jobs which can be performed by machine i within target performance time T. The bundles in $C(i,T)$ will determine the edges in the following bipartite hypergraphs associated to problem instances with a given target T.

Definition 4.4. Given an instance of the Santa Claus problem and target allocation T we define a hypergraph $\mathcal{H}_T := (P \cup R, \mathcal{E}_T)$ where $\mathcal{E}_T := \{\{i\} \cup S : i \in P \text{ and } S \in C(i,T)\}$.

Definition 4.5. Given an instance of the machine scheduling problem and target makespan T we define a hypergraph $\mathcal{H}'_T := (M \cup J, \mathcal{E}'_T)$ where $\mathcal{E}'_T := \{\{i\} \cup S : i \in M \text{ and } S \in C(i,T)\}$.

For the Santa Claus problem, finding bundles $S_i \in C(i,T)$ to allocate to each player to meet our target T corresponds to the problem of finding a P-saturating matching in the bipartite hypergraph $\mathcal{H}_T$.

To meet a target makespan T, the machine scheduling problem is the task of allocating a bundle of jobs S_i to each machine i so that each machine can perform the jobs in its bundle within time T and with $\bigcup_{i \in M} S_i = J$ ensuring all jobs get assigned. This corresponds to the problem of finding an edge cover for J in the bipartite hypergraph $\mathcal{H}'_T$ such that no two edges intersect in M (vertices of M unsaturated by such a cover get an empty bundle of jobs).

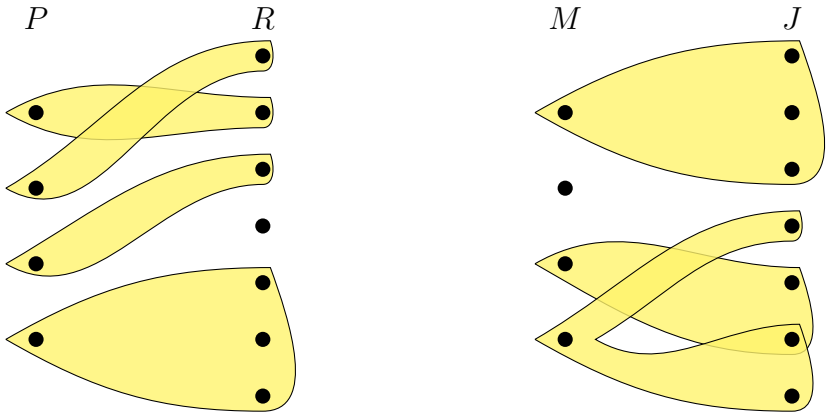

Figure 4.1: Edges are indicated by yellow regions marked with solid line borders. On the left is a selection of edges which is a matching saturating P, as desired among edges of hypergraph $\mathcal{H}_T$ to meet target allocation T. On the right is a selection of edges that cover J and are disjoint in M, as desired among edges of $\mathcal{H}'_T$ to satisfy target makespan T.

So far it has been results and ideas related to A-saturating (A,X) bipartite hypergraph matchings that have applied naturally to the Santa Claus problem (see [4] for example). It is hoped that interesting results, perhaps related to edge covers, on bipartite hypergraphs may be underlying for the machine scheduling problem also. We attempt to investigate some of these related notions to edge covers on bipartite hypergraphs in Chapter 8.

Definition 4.6. Given a bipartite hypergraph $\mathcal{H} = (A \cup X, \mathcal{E})$, for $B \subset A$ we define $\mathcal{H}_B$ to be the hypergraph with vertices X and edges $\mathcal{F} := \{E \cap X : E \in \mathcal{E} \text{ and } E \cap B \neq \emptyset\}$. It is the hypergraph formed on X by taking the edges which hit B and deleting from each edge that single vertex hitting B.

A ubiquitous result on maximum matchings in bipartite graphs is Hall's Theorem.

Theorem 4.7 (Hall's Theorem)**.** *Suppose G is a bipartite graph with bipartition (U, V). There exists a matching in G saturating U if and only if for every $S \subset U$ we have the set of neighbours to vertices in S denoted by $N(S)$ satisfying $|N(S)| \geq |S|$.*

A sufficient condition for saturating the A-side of a bipartite hypergraph is given by the following result that generalizes Hall's Theorem 4.7.

Theorem 4.8 (Haxell's Theorem [15])**.** *Let $\mathcal{H} = (A \cup X, \mathcal{E})$ be a bipartite hypergraph with edges containing at most r vertices, such that for every $B \subset A$, every transversal of $\mathcal{H}_B$ has size greater than $(2r-3)(|B|-1)$. Then $\mathcal{H}$ admits an A-saturating matching.*

Taking $r = 1$ yields a statement equivalent to the non-trivial direction of Hall's Theorem. The proof [15] of this result inspired the ideas in the original proof showing the integrality gap of the restricted Santa Claus CLP (3.1)-(3.4) is at most 4 [4].

The proof of Haxell's Theorem, proceeds by the contrapositive to show that having a max-matching not saturating A implies the existence of a small transversal for some $\mathcal{H}_B$. The proof uses a local search starting with an A-vertex unsaturated by a max-matching and proceeds to build up a tree of adding edges and blocking edges which come from the current max matching. The max matching is made to update upon addition of each adding edge, but to retain all blocking edges that appear previously in the tree. When the procedure for building the tree is finished, the transversal on $\mathcal{H}_B$, for B the set of A-vertices appearing among edges in the tree, is then formed by the set of X-vertices in (edges of) the tree. In fact this proof shows a particular structure for the transversal it produces. This structure shall be made evident in another version of Haxell's Theorem we shall describe shortly.

4.2 Independent Transversals and Estimation for Restricted Santa Claus

In this section we will give an alternative proof that the CLP has integrality gap at most 4 based on a version of Haxell's Theorem 4.8 which concerns independent transversals in vertex partitioned graphs. First we introduce the relevant definitions (as laid out in a paper by Graf and Haxell [14]).

For a partition of vertices of a graph $G = (V, E)$ into $\{V_i : i \in I\}$ and a set of some of these vertex classes B, we use G_B to denote the subgraph of G induced on the union of the vertex classes in B.

Definition 4.9. For a graph $G = (V, E)$ and a set of vertices $D \subset V$, we say D *(totally) dominates* G provided that for every vertex in V there exists an edge incident with it and some vertex of D.

Note that there is another weaker notion of domination, which we do not discuss here.

Definition 4.10. An *independent transversal* of a graph $G = (V, E)$ with respect to a vertex partition $\{V_i : i \in I\}$, is an independent set (or coclique) of vertices $\{v_i : i \in I\}$ in G, with $v_i \in V_i$ for all i.

Definition 4.11. A *constellation* K, for a set of vertex classes B in a vertex partitioned graph $G = (V, E)$, is an induced subgraph of G_B that satisfies the following properties. Its components are stars $K_{1,n}$ for some $n \geq 1$, with one vertex designated as the centre (the central vertex for $n \geq 2$ and one of the vertices is chosen as the centre for $K_{1,1} = K_2$) and the rest as leaves. Finally, the set of leaves in K form an independent transversal for $|B| - 1$ of the vertex classes.

Note that there are $|B| - 1$ leaves of a constellation K, each in a different vertex class of B. It follows that $|V(K)| \leq 2(|B| - 1)$ since each component has at least as many leaves as the one central vertex. An example constellation is pictured below in Figure 4.2. Note that a constellation induces a rooted tree structure on vertex classes of B with the unique vertex class not containing a leaf serving as the root.

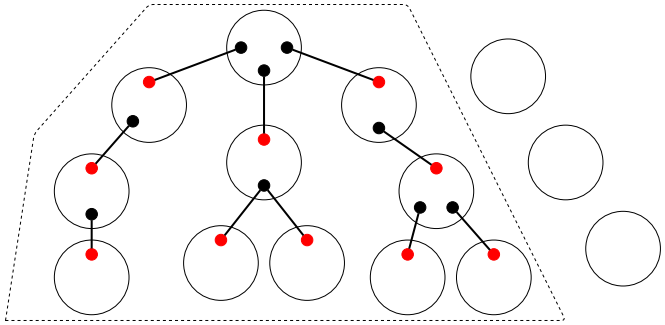

Figure 4.2: A constellation K for the set B of vertex classes contained within the dotted border. Vertex classes are indicated by circles. The centres of the stars appear in black and the leaves appear in red. The rest of the graph G is omitted.

We have the following result which follows from the proof of Haxell's Theorem [15] and was explicitly given in a similar form in a paper by Aharoni, Berger and Ziv [1]. An algorithmic version of this result was given in a paper by Graf and Haxell [14].

Theorem 4.12. *Let $G = (V, E)$ be a graph partitioned into vertex classes. If there does not exist an independent transversal in G with respect to this partition then there exists a subset of vertex classes B and a constellation K for B such that $V(K)$ dominates G_B.*

We omit the proof and use this result combined with the dual of the CLP to provide a novel proof that the CLP for an instance of the restricted Santa Claus problem has an

integrality gap of at most 4. Let $\mathcal{E}_a$ denote the hyperedges in $\mathcal{E}$ hitting vertex $a \in A$. This Theorem 4.12 relates to the original statement of Haxell's Theorem 4.8, by considering a vertex partitioned graph G associated to a bipartite hypergraph $\mathcal{H} = (A \cup X, \mathcal{E})$, where vertices of G correspond to hyperedges $\mathcal{E}$ which are partitioned into classes $\mathcal{E}_a$ for each $a \in A$, with two hyperedges adjacent in G if and only if they intersect in X.

Theorem 4.13. *The integrality gap of the restricted Santa Claus CLP is at most 4.*

Proof. We recall the CLP (3.1)-(3.4) and its dual (3.5)-(3.7) for the Santa Claus problem. By scaling of resource values it suffices to show given that the CLP has no integral solution for $T = 1$, that the CLP is infeasible for $T = 4$. To certify this we exhibit a solution to the dual of the CLP for $T = 4$ with positive objective value (hence by primal minimization problem with constant objective of 0 this certifies infeasibility by weak duality).

Let $G = (V, E)$ have vertices $\{i, S\}$ for every player i and every minimal (by inclusion) bundle S of resources that provides value at least 1 to player i. Edges are put between any vertices that share a resource in their bundles. Vertices are partitioned into $V = \{V_i : i \in P\}$ by the player they correspond to. An independent transversal of G with respect to this partition corresponds to choosing disjoint bundles of resources of value at least 1 for all players, hence finding an integral solution to the Santa Claus CLP for $T = 1$. So by assumption no such independent transversal exists in G with respect to this partition. Applying Theorem 4.12 we obtain a subset of the vertex classes B and a constellation K for B whose vertices dominate G_B. We identify B with the corresponding set of players we are partitioning by. We note that since K is induced, any two of its non-adjacent vertices share no resource. Let U be the set of all resources that appear among the vertices in K. We define the dual variables, see (3.5)-(3.7), based on B and U as follows:

$$
\begin{aligned}
y_i &:= 0 && \forall i \in P \setminus B \\
y_i &:= 3 && \forall i \in B \\
z_j &:= 0 && \forall j \in R \setminus U \\
z_j &:= 3 && \forall j \in U \text{ with } p_j \geq 1 \\
z_j &:= p_j && \forall j \in U \text{ with } p_j < 1.
\end{aligned}
$$

Note that (y, z) clearly satisfies the non-negativity constraints and we shall proceed to verify that it satisfies the other constraints and gives positive objective value.

Let S be a bundle assignable to a player $i \in B$ of value at least 1 such that $U \cap S = \emptyset$, then $\{i, S\}$ would be a vertex in G_B not adjacent to any vertex of K, contradicting K dominating G_B. Therefore when S is a bundle assignable to player $i \in B$ of value at least 1 we have $U \cap S \neq \emptyset$. Thus for any $i \in B$ and $S \in C(i, 4)$ we have that the total value of resources in the intersection of S and U is at least 3, as if it were less than 3 deleting the intersection from S would produce a disjoint bundle from U of value at least 1 assignable to player i. Because all resources in U are assigned either a dual variable entry of 3 or at least its inherent value, we have that for all $i \in B$ and $S \in C(i, 4)$: $y_i = 3 \leq \sum_{j \in S} z_j$. By our setting y to be 0 outside of B, the other feasibility constraints hold also. So it remains to verify that the objective is positive.

We have $\sum_{i\in P} y_i = 3|B|$. Now $\sum_{j\in R} z_j$ is equal, by resource disjointness of components in K and z zero outside of U, to the sum of the total z-value on resources in each of the components/stars of K. Consider any star of our constellation K and let its centre be given by $v = (i_0, S_0)$.

Case 1: S_0 contains a resource of value at least 1.
Due to the minimality of S_0, S_0 must consist of a single resource. Any of the neighbours of v in K must also share, and hence by minimality entirely consist of, this one resource. Thus the total z-value of resources over all of this component of K is 3. Components in K consist of at least 2 vertices (in fact here it must be exactly 2) so the z-value of this component per vertex is at most $\frac{3}{2}$.

Case 2: S_0 contains only resources of value less than 1.
Since S_0 is minimal, the inherent value of resources in S_0 is less than 2 as otherwise dropping any resource would keep it satisfying with value at least 1. Let $v' = (i', S')$ be a neighbour to v in its component of K. From minimality of S', the value of $S' \setminus S_0$ must be less than 1, as otherwise we could drop from S' its non-empty (by adjacency of v and v') intersection with S_0. Note this tells us that all resources appearing in this star have value less than 1, hence all having value equal to their corresponding dual variable z-values. Let n be the number of vertices in this star, so the star consists of v and its $n-1$ neighbours. We conclude that the total z-value of resources in this component is at most $2+(n-1) = n+1$. So by n at least 2, this gives us that the z-value of this component per vertex is at most $\frac{3}{2}$.

Therefore the total z-value over all of R, which is the total taken over all components of K, is at most $\frac{3}{2}|V(K)|$. Now $|V(K)| \leq 2(|B|-1) < 2|B|$, so we have:

$$\sum_{j\in R} z_j < \frac{3}{2}(2|B|) = 3|B| = \sum_{i\in P} y_i.$$

Therefore $\sum_{i\in P} y_i - \sum_{j\in R} z_j > 0$ so the feasible dual solution attains a positive objective value. This certifies the infeasibility of the CLP for $T = 4$, thus showing that the integrality gap for the CLP is at most 4. □

This result shows that (see Theorem 3.6) using the CLP we have a polytime method for estimating the optimal max-min of the restricted Santa Claus problem within any factor greater than 4, leaving a gap between this approximation ratio of 4 (which has been improved somewhat [9] to a best known factor of $4 - \frac{5}{26}$) and 2 where we have an NP-hard complexity result for estimation of the max-min. A constant factor approximation algorithm for the restricted Santa Claus has also been obtained in work by Annamalai [2]. This has been later improved, in a paper by Davies, Rothvoss and Zhang to $4+\epsilon$ [11], the factor of estimation shown here. This paper [11], which we discuss in Chapter 6, generalizes the Santa Claus problem to a setting where one wants an allocation satisfying some basis of a matroid rather than a fixed set of players, and applies an algorithm developed in that more general setting to the Santa Claus problem. The CLP itself is not known to have integrality gap worse than 2, so it is hoped that the CLP fully captures the difficulty of approximation for the restricted Santa Claus problem.

A concrete example in which the restricted Santa Claus CLP has integrality gap of 2 is as follows. Consider the following problem instance for the restricted Santa Claus problem: We are given $P = \{1, 2, 3, 4\}$ and $R = \{A, B, C, D, E, F\}$ with resource values $v_A = v_B = 2$ and $v_C = v_D = v_E = v_F = 1$. We also have allowable assignments determined by the following:

$$\begin{aligned}
&\text{Player 1 may be assigned resources } A, C, D\\
&\text{Player 2 may be assigned resources } A, E, F\\
&\text{Player 3 may be assigned resources } B, C, F\\
&\text{Player 4 may be assigned resources } B, D, E.
\end{aligned}$$

We claim the max-min for any allocation of bundles to the above problem is at most 1. Suppose for contradiction there was an allocation giving value at least 2 to all players. Since only one of players 1 and 2 may be assigned resource A, one of them must receive both of the resources it gives value 1. Thus either $\{C, D\}$ or $\{E, F\}$ must be resources allocated to players among 1 and 2. By identical reasoning we conclude either $\{C, F\}$ or $\{D, E\}$ must be resources allocated to players among 3 and 4. Since any choice among $\{C, F\}$ or $\{D, E\}$ intersects both $\{C, D\}$ and $\{E, F\}$, this tells us a resource is allocated both to a player among $\{1, 2\}$ and $\{3, 4\}$ contradicting disjointness of bundles. So by the max-min integer given this integer data, we have that the max-min is at most 1.

On the other hand we have a fractional allocation for the Santa Claus CLP for $T = 2$, given by for each player half-allocating a bundle consisting of the 2-value resource it may be assigned, and half-allocating a bundle consisting of both of the 1-value resources it may be assigned. So this is a concrete example in which the Santa Claus CLP has integrality gap of 2. With essentially no alteration, this gives an example for the corresponding machine scheduling CLP with largest known integrality gap $\frac{3}{2}$. We next introduce this machine scheduling CLP and discuss its applications to the restricted machine scheduling problem.

Chapter 5

Restricted Machine Scheduling Estimation

Here we give the CLP and its dual for the machine scheduling problem with target makespan T. Recall that $C(i,T)$ denotes the set of configurations for machine i and target T, where a configuration is a set of jobs S with $\sum_{j\in S} p_{ij} \leq T$.

Machine Scheduling CLP for target T

$$\max \quad 0 \tag{5.1}$$

$$\sum_{S\in C(i,T)} x_{i,S} \leq 1 \qquad \forall i \in M \tag{5.2}$$

$$\sum_{i,S:j\in S, S\in C(i,T)} x_{i,S} \geq 1 \qquad \forall j \in J \tag{5.3}$$

$$x_{i,S} \geq 0 \qquad \forall i \in M, \forall S \in C(i,T). \tag{5.4}$$

Machine Scheduling Dual CLP for target T

$$\min \quad \sum_{i\in M} y_i - \sum_{j\in J} z_j \tag{5.5}$$

$$y_i \geq \sum_{j\in S} z_j \qquad \forall i \in M, S \in C(i,T) \tag{5.6}$$

$$y_i, z_j \geq 0 \qquad \forall i \in M, \forall j \in J. \tag{5.7}$$

The primal CLP is a relaxation of the task of assigning bundles of jobs to machines that do not have total completion time exceeding the target makespan T. Note that by similar reasoning as in the Santa Claus CLP we have that a negative objective solution to the dual certifies the infeasibility of the primal for the same target T. Also, as in the Santa Claus CLP (see Theorem 3.6), determining the integrality gap of the CLP for a class of problem instances shows that the optimal makespan may be estimated in polytime for those instances within any factor greater than the corresponding integrality gap. There

is also an equality version of this CLP, which we will later apply to the analysis of *graph balancing* variants of the machine scheduling problem.

The CLP and its dual have also been useful in the analysis of machine scheduling problem for the restricted case. In a paper due to Svensson [28], by analysing the integrality gap of the CLP it was shown that the makespan may be estimated in polytime up to a factor arbitrarily close to $\frac{33}{17}$. This result was improved to a factor of $\frac{11}{6}$ with a simplified argument in a paper by Jansen and Rohwedder [19] which we shall discuss in more detail below.

These results have yet to be made into a polytime algorithm for finding an assignment with that same approximation ratio for the optimal makespan, so the general case 2-approximation algorithm discussed earlier [23] remains the best known for the restricted case.

We shall proceed, in the current and following sections, to describe in detail the paper of Jansen and Rohwedder [19], which gives the $\frac{11}{6}$ estimation result by analysis of the integrality gap of the restricted machine scheduling CLP.

Note that in the restricted case we are now considering, we have fixed job sizes p_j for each job. Recall that $\Gamma(j)$ denotes the set of machines capable of performing job j. The set of configurations for player i, $C(i,T)$, is now given by the set of all bundles of jobs assignable to machine i of total size at most T. Otherwise, the CLP and its dual (5.1)-(5.7) are the same as written above.

Suppose we are given an instance of the restricted machine scheduling problem. By scaling job sizes we can set $T = 1$ as the optimal (least) value of T for which the CLP is feasible. Note that all job sizes p_j are then at most 1. To show the integrality gap is at most $1 + r$ for some $r \in (0, 1]$, it suffices to produce a job assignment of makespan at most $1 + r$.

5.1 A Preliminary Analysis

To illustrate some of the basic ideas in the general restricted case, we shall first assume that all job sizes are small, that is $p_j \leq r$ for all jobs j and some fixed $r \in (0, 1]$. It will be convenient to use notation $f(U)$ to represent the sum $\sum_{j \in U} f(j)$ for $U \subset J$ and f a real valued function on the jobs (such as the function p giving job times). We will introduce some relevant definitions (closely related to those provided in [28, 19]) and describe an algorithm which produces a desired job assignment of makespan at most $1 + r$. To show that our algorithm doesn't get stuck, a negative objective dual solution is constructed based on the progress of our algorithm that certifies the infeasibility of the primal CLP for $T = 1$, giving a contradiction. The dual solution is constructed based on a *blocking tree* $\mathcal{T}$ that prevents the assignment of an additional job j_{new} to a *partial schedule* σ. The variables $\mathcal{T}$, j_{new} and σ are the variables updated during the running of the algorithm.

Definition 5.1. A *partial schedule* is a function $\phi : J \to M \cup \{\perp\}$ such that for all $j \in J$ we have $\phi(j) \in \Gamma(j)$ or $\phi(j) = \perp$. Here $\perp$ is a special symbol indicating *unassigned* jobs, jobs not taking that value are said to be *assigned* by ϕ.

A partial schedule ϕ is *valid* when for all $i \in M$ we have $p(\phi^{-1}(i)) \leq 1+r$. This captures the notion of not having already, in the partial assignment, overworked any machine to violate the target makespan $1 + r$. Throughout the algorithm the partial schedule σ will be kept valid and this is the only partial schedule that the algorithm will consider.

Definition 5.2. A *move* is a pair (j, i) for job j and machine i where $i \neq \sigma(j)$ and $i \in \Gamma(j)$. A move (j, i) is said to be *valid* when $p(\sigma^{-1}(i)) + p_j \leq 1 + r$ and *invalid* otherwise.

When a move (j, i) is valid updating σ by $\sigma(j) \leftarrow i$ maintains σ as a valid partial schedule, so only executing valid moves to update σ during our algorithm ensures σ remains a valid partial schedule.

Definition 5.3. A *blocking tree* $\mathcal{U}$ is a pair consisting of: (1) a rooted tree on a set of vertices having a special root vertex designated as ROOT, with other vertices being moves and (2) a list which orders the moves that appear as vertices in $\mathcal{U}$.

In an abuse of notation we will frequently identify $\mathcal{U}$ with its corresponding list of moves, leaving the tree structure implicit. We let $M(\mathcal{U})$ denote the set of machines appearing among the moves in $\mathcal{U}$.

The blocking tree that will be updated during our algorithm is $\mathcal{T}$, which along with the current partial assignment σ, determines which *potential moves* we are interested in considering adding to our blocking tree. We call elements of $M(\mathcal{T})$ *blocking machines* and say other machines are *not blocking*. Blocking machines will be made to appear uniquely in the blocking tree $\mathcal{T}$, while the same job can appear in many of the moves of $\mathcal{T}$.

Definition 5.4. We define a set of jobs $J(\mathcal{T})$ associated to our blocking tree $\mathcal{T}$ by

$$J(\mathcal{T}) := \{j_{new}\} \cup \bigcup_{i \in M(\mathcal{T})} \sigma^{-1}(i).$$

Note that $J(\mathcal{T})$ is also dependent on other variables j_{new} and σ. Here $J(\mathcal{T})$ will be the set of jobs that the blocking tree $\mathcal{T}$ indicates we should try to move to make room for assignment of the new job j_{new}.

Definition 5.5. A *potential move* is a move (j, i) such that $j \in J(\mathcal{T})$ and $i \in M \setminus M(\mathcal{T})$.

Potential moves are trying to move a job, which is either the new job to be assigned, j_{new}, or a job currently assigned to a blocking machine, towards a machine that is not blocking. Potential moves also have an associated *value* and corresponding *priority*, determining which potential moves our algorithm should consider. Note that as soon as a potential move (j, i) is added to $\mathcal{T}$ it ceases to be a potential move, as subsequently $i \in M(\mathcal{T})$.

Definition 5.6. The *value* of a potential move (j, i) is $\mathrm{Val}(j, i) := |\sigma^{-1}(i)|$, the number of jobs currently assigned to the machine i we are trying to move j into.

Potential moves are then given *priority* so that a move (j, i) has higher priority for lesser $\text{Val}(j, i)$. Potential moves of same value have the same priority. In the general case there will be different types of potential moves, and the value of a move will then be a pair of integers (the first of which denotes the type) to be prioritized in lexicographically decreasing order.

Description of $\mathcal{T}$: We will update a blocking tree $\mathcal{T}$ during our algorithm, building it up from its initialization as the trivial tree on ROOT whenever we try to assign a new job j_{new}. Newly added moves to $\mathcal{T}$ are appended to the end of the list of moves, so the list of moves is ordered by the order in which the algorithm adds them to $\mathcal{T}$. We let $\ell \in \mathbb{N}$ be the number of moves appearing as vertices in $\mathcal{T}$, so $\mathcal{T} = (m_1, \ldots, m_\ell)$, where m_k is the k^{th} move that was added to $\mathcal{T}$ (among those remaining as vertices in $\mathcal{T}$).

Definition 5.7. Let k be such that $0 \leq k \leq \ell$, we define $\mathcal{T}^k := (m_1, \ldots, m_k)$ to be the blocking tree which inherits the tree structure from $\mathcal{T}$ on its first k moves and ROOT, with moves listed in same order as they appear in $\mathcal{T}$.

Updating $\mathcal{T}, \sigma$ and j_{new}: We initially set σ as the empty assignment which leaves all jobs unassigned, that is $\sigma(j) = \perp$ for all jobs j. We initially set j_{new} to be any job and $\mathcal{T}$ as the trivial blocking tree on ROOT with empty list of moves. When there is not a valid move in $\mathcal{T}$, we extend the blocking tree by finding a potential move (j_0, i_0) of highest priority.

1. If $j_0 = j_{new}$ we give this move (j_0, i_0) parent ROOT in the tree structure and append it to the list of moves for $\mathcal{T}$.

2. Otherwise $i = \sigma(j_0)$ for some $i \in M(\mathcal{T})$. Let m_k be the first (and only) move in $\mathcal{T}$ with machine $i = \sigma(j_0)$. We then give move (j_0, i_0) parent m_k in the tree structure and append it to the list of moves for $\mathcal{T}$.

When there is a valid move among the moves in $\mathcal{T}$, we pick one (j_0, i_0) and update σ by setting $\sigma(j_0) \leftarrow i_0$. We update $\mathcal{T}$ by deleting some moves as follows:

1. If $j_0 = j_{new}$ we have assigned j_{new}, so we reset $\mathcal{T}$ to the trivial blocking tree on ROOT (with empty list), and update j_{new} to some other unassigned job. If there is no unassigned job remaining the algorithm terminates, outputting σ.

2. Otherwise (j_0, i_0) has parent move m_k in $\mathcal{T}$. We then update $\mathcal{T}$ by $\mathcal{T} \leftarrow \mathcal{T}^{k-1}$, deleting all moves from parent m_k onwards. The parent m_k will be such that its corresponding machine i had $i = \sigma(j_0)$ prior this update of σ.

We then describe the algorithm for producing an assignment of jobs with makespan at most $1 + r$ in pseudocode as follows:

```
function Special Case Build Job Assignment
    σ is set to the empty assignment, a constant map to ⊥
    while there is job unassigned by σ do
        j_new is set to an unassigned job
        𝒯 is set to trivial blocking tree on single node ROOT
        The list of moves associated with 𝒯 is set to be empty
        while σ(j_new) =⊥ do
            if there exists a move (j, i) in 𝒯 which is valid then
                if j = j_new then
                    σ(j_new) ← i
                else
                    σ(j) ← i
                    𝒯 ← 𝒯^{k−1} and delete moves from m_k and beyond in list of moves,
    where m_k is the parent of (j, i)
                end if
            else if there exists a potential move then
                Find (j, i) a highest priority potential move
                if j = j_new then
                    Update 𝒯 by adding (j, i) with parent ROOT and append (j, i) to
    the list of moves
                else
                    Find (first) move m_k on machine σ(j) in 𝒯
                    Update 𝒯 by adding (j, i) with parent m_k and append (j, i) to the
    list of moves
                end if
            else
                return "FAIL" and end function
            end if
        end while
    end while
    return σ
end function
```

A machine i cannot appear twice among the moves of $\mathcal{T}$, since after its introduction into $\mathcal{T}$ subsequent potential moves considered for addition to $\mathcal{T}$ cannot use machine i. Also note that for as long as machine i continuously remains a blocking machine in $M(\mathcal{T})$, $\sigma^{-1}(i)$ is unchanging. This follows from: (a) performing a valid move which moves away a job from $\sigma^{-1}(i)$ deletes from $\mathcal{T}$ moves containing i and (b) moves added into $\mathcal{T}$ after the introduction of i into $M(\mathcal{T})$ cannot use machine i, so when executed do not add jobs to $\sigma^{-1}(i)$. So in particular we have that pairs (j, i) in $\mathcal{T}$ indeed remain moves.

We shall proceed to argue that this algorithm terminates and that it produces a valid partial job assignment σ with no jobs left unassigned, hence a job assignment with makespan at most $1 + r$ as desired.

Only valid moves are made to σ and the outer loop runs until we return "FAIL" or have assigned all jobs. So provided the algorithm returns σ it produces a job assignment

with makespan at most $1 + r$. To show correctness of the algorithm we need to verify termination and that "FAIL" is never output.

Theorem 5.8. *The algorithm does not output "FAIL" and thus when terminating produces a job assignment of makespan at most* $1 + r$.

Proof. Suppose for contradiction that we return "FAIL" and let $j_{new}, \mathcal{T}$ and σ be the variables at the start of the inner loop iteration where this occurs. We must have no valid moves in $\mathcal{T}$ and no potential moves. We construct dual variables for the Dual CLP (5.5)-(5.7) with $T = 1$ as follows:

$$z_j := p_j \qquad \forall j \in J(\mathcal{T}) \tag{5.8}$$

$$z_j := 0 \qquad \text{otherwise.} \tag{5.9}$$

$$y_i := z(\sigma^{-1}(i)) \qquad \forall i \in M. \tag{5.10}$$

Claim 1: (y, z) is feasible for the machine scheduling Dual CLP (5.5)-(5.7) with $T = 1$. Clearly this candidate dual solution (y, z) satisfies non-negativity constraints. To check other constraints, we let $i \in M$ and $C \in C(i, 1)$, and the corresponding inequality to verify is $y_i \geq z(C)$.

Case 1: $i \in M(\mathcal{T})$.
Since C is a configuration for target $T = 1$, we have $p(C) \leq 1$. We also have $z_j \leq p_j$ for all jobs, so $z(C) \leq p(C)$. Therefore it suffices to verify $y_i \geq 1$.

Since $i \in M(\mathcal{T})$ there is a move (j, i) in $\mathcal{T}$, which must not be valid. Therefore $p(\sigma^{-1}(i)) + p_j > 1 + r$. From assumption $p_j \leq r$, so combining yields $p(\sigma^{-1}(i)) > 1$. Note that $\sigma^{-1}(i) \subset J(\mathcal{T})$, therefore $y_i = z(\sigma^{-1}(i)) = p(\sigma^{-1}(i)) > 1$. Thus inequality $y_i \geq z(C)$ is satisfied.

Case 2: $i \in M \setminus M(\mathcal{T})$.
If $j \in C \cap J(\mathcal{T})$ then j is assignable to i and (j, i) is thus a potential move, giving a contradiction. Therefore $C \cap J(\mathcal{T}) = \emptyset$ and $z(C) = 0$. So the constraint holds as y_i is non-negative (in fact it is 0 in this case).

Thus in all cases for constraints, we have verified that (y, z) is satisfying. So (y, z) is a feasible Dual CLP solution, verifying our claim.

Claim 2: (y, z) has negative objective $\sum_{i \in M} y_i - \sum_{j \in J} z_j$.
Note that $j_{new} \notin \sigma^{-1}(i)$ for any $i \in M$, and therefore:

$$\sum_{j \in J} z_j \geq z_{j_{new}} + \sum_{i \in M} z(\sigma^{-1}(i)). \tag{5.11}$$

Since $j_{new} \in J(\mathcal{T})$ we have $z_{j_{new}} > 0$, and by definition $\sum_{i \in M} z(\sigma^{-1}(i)) = \sum_{i \in M} y_i$. Applying this to (5.11) yields $\sum_{i \in M} y_i - \sum_{j \in J} z_j \leq -z_{j_{new}} < 0$ as desired.

Combining claims, we have that (y, z) is a feasible solution with negative objective to the Dual CLP for $T = 1$, contradicting feasibility of the CLP for $T = 1$. Therefore our algorithm does not output "FAIL" and when terminating produces a job assignment of makespan at most $1 + r$. □

Theorem 5.9. *The Special Case Build Job Assignment algorithm terminates.*

Proof. To demonstrate the termination of the algorithm we associate at each stage a *signature vector* $\text{Sig}(\mathcal{T}) := (s_1, \ldots, s_\ell, \infty)$ to our blocking tree $\mathcal{T} = (m_1, \ldots, m_\ell)$. Here for each k, s_k is the value $\text{Val}(m_k)$ that m_k had as a potential move just prior to the addition of m_k to $\mathcal{T}$.

The number of moves in $\mathcal{T}$ is bounded by $|M|$ since every machine can appear at most once in the moves of $\mathcal{T}$. So signature vectors have bounded length. Furthermore, since any set of jobs has size at most $|J| = n$, we have a bounded number of possible values for any potential move. Therefore we have a bounded (but exponential) number of possible signature vector values that can be taken during our algorithm.

Note that any line of algorithm can be executed by exhaustive search in finite time, and that the outer loop is entered at most n times. Therefore to show termination it suffices to show we can only update $\mathcal{T}$ and σ finitely many times in trying to assign fixed job j_{new}. Each valid move executed reduces the size of the current list of moves for $\mathcal{T}$. So between the addition of any two consecutively added moves to $\mathcal{T}$ during the algorithm, only finitely many valid moves can be made. Combining this with the bounded number of possible signature vectors that can occur, it suffices to show that the signature vector after adding a move to $\mathcal{T}$ is (strictly) less lexicographically than the signature vector after adding the previous move.

Let $\mathcal{T}_0 = (m_1^0, \ldots, m_\ell^0)$ be the blocking tree $\mathcal{T}$ after addition of the previous move m_ℓ^0 and let $\mathcal{T}_1 = (m_1^0, \ldots, m_k^0, m_{add})$ for some $k \leq \ell$ be the blocking tree $\mathcal{T}$ after addition of next move m_{add}. Let s_{add} be the value of potential move m_{add} when it was added to $\mathcal{T}$. Then we have associated signature vectors $V_0 = (s_1^0, \ldots, s_\ell^0, \infty)$ and $V_1 = (s_1^0, \ldots, s_k^0, s_{add}, \infty)$ for $\mathcal{T}_0$ and $\mathcal{T}_1$ respectively.

Case 1: $k = \ell$.
Then no valid moves were performed between addition of these moves, and signature vectors V_0 and V_1 are identical on first ℓ entries. The $\ell + 1^{st}$ entry is ∞ for V_0 and is $s_{add} < \infty$ for V_1, therefore V_1 is less than V_0 lexicographically.

Case 2: $k < \ell$.
Let $m_{k+1}^0 = (j, i)$. Let $m_f = (j_f, i_f)$ be the final valid move made between addition of moves m_ℓ^0 and m_{add} to $\mathcal{T}$. Updating $\mathcal{T}$ after executing m_f deleted moves from m_{k+1}^0 onwards. So m_{k+1}^0 was the parent of m_f, which implies that j_f was in $\sigma^{-1}(i)$ while $m_{k+1}^0 = (j, i)$ remained in $\mathcal{T}$. So executing m_f reduces $|\sigma^{-1}(i)|$. Since $\sigma^{-1}(i)$ was unchanging while $m_{k+1}^0 = (j, i)$ remained in $\mathcal{T}$, we have that $|\sigma^{-1}(i)|$ is smaller after making this final valid move m_f than s_{k+1}^0, the size of $\sigma^{-1}(i)$ just prior to the introduction of move m_{k+1}^0 to $\mathcal{T}$.

Consider our algorithm at stage with $\mathcal{T} = (m_1^0, \ldots, m_k^0)$ after making this last valid move m_f. Now, upon discovering no remaining valid moves in $\mathcal{T}$, our algorithm seeks the

highest priority potential move. When $m^0_{k+1} = (j, i)$ was introduced earlier, (j, i) was a potential move on the same $\mathcal{T}$. We have that σ^{-1} has remained unchanged for machines in $M(\mathcal{T})$ since $(m^0_1, \ldots, m^0_k)$ have not been deleted. Therefore (j, i) is once again a potential move, and its new value $\mathrm{Val}(j, i) = |\sigma^{-1}(i)|$ has decreased from its earlier value s^0_{k+1}. Therefore the value of a highest priority potential move s_{add} is strictly less than s^0_{k+1}. So V_0 and V_1 are identical on first k entries and V_1 is less than V_0 on $k + 1^{th}$ entry, therefore the signature decreases lexicographically.

Thus in any case we see a decrease in the signature vector with V_1 less than V_0. Therefore the algorithm terminates. □

Combining our results on output and termination of the algorithm, we obtain the following results.

Theorem 5.10. *Suppose we are given restricted machine scheduling problem with $T = 1$ the optimal T for which the CLP is feasible, and all job sizes $p_j \leq r$. Then the algorithm Special Case Build Job Assignment produces a job assignment with makespan at most $1 + r$.* □

Theorem 5.11. *Suppose we are given restricted machine scheduling problem with $T = 1$ the optimal T for which the CLP is feasible, and all job sizes $p_j \leq r$. The integrality gap for the corresponding CLP is at most $1 + r$.* □

Theorem 5.12. *We may estimate in polytime the optimal makespan within any factor greater than $1 + r$, for the class of instances of the restricted machine scheduling problem which have all job sizes at most r times the optimal T for which the corresponding CLP is feasible.* □

Note that any job size is at most the optimal T for which the CLP is feasible, so taking $r = 1$ above gives an alternative proof (as opposed to applying the 2-approximation algorithm) that the optimal makespan may be estimated to within any factor greater than 2. It does not yield the same approximation factor as the construction of the job assignment in this algorithm was not shown to be polytime.

This $1+r$ estimation result for the restricted case is not novel in of itself and does not in general improve on the 2-approximation algorithm guarantee provided by Lenstra et al. [23]. Indeed, under the weaker assumption that the maximum (non-infinite) performance time of any job on any machine is at most r times the optimal makespan, the much earlier work of Lenstra et al. [23] gives a $1 + r$ approximation algorithm. However the techniques here can, with some care and technical complications, be made to give an improved estimation result in general [28, 19], as we shall see in the following section.

5.2 The General Case for 11/6 Estimation

Here we give the general $\frac{11}{6}$ estimation result for the restricted machine scheduling problem. Let $r = \frac{5}{6}$ so that $\frac{11}{6} = 1 + r$. The choice of $r = \frac{5}{6}$ will be the least possible for this analysis

to go through as we shall require the inequality $(1+r) - \frac{1}{2} - 2(1-r) \geq 1$ to be satisfied (it is the underlying inequality in the analysis of the first case for feasibility in the proof of Theorem 5.25). As before, it suffices to show that given that the CLP is feasible for $T = 1$ that there exists a job assignment of makespan at most $1 + r$. So we assume that the CLP is feasible for $T = 1$ and proceed to describe an algorithm which builds a job assignment of makespan at most the target $1 + r$. Again, note that the CLP being feasible for $T = 1$ ensures that all job sizes are at most 1.

As in the previous algorithm, the variables for our algorithm will be *partial schedule* σ, *blocking tree* $\mathcal{T}$ and job j_{new}. It will be convenient to label jobs as $J := \{1, \ldots, n\}$, providing a total order on jobs, and we will choose our labelling so that jobs have sizes coming in weakly increasing order. The definitions and results presented will follow closely the paper of Jansen and Rohwedder [19], which itself adapts many similar notions from the earlier paper [28] due to Svensson.

Partial schedules and the condition for *validity* are defined exactly as before (see Definition 5.1), taking $r = \frac{5}{6}$. *Moves* and *valid* moves are also defined as before in Definition 5.2, taking $r = \frac{5}{6}$.

Since we have no assumption on the sizes of jobs, we make a division of job sizes into *big* jobs and *small* jobs. This division will result in differing types of *potential moves* and consequently different types of vertices in our blocking tree.

Definition 5.13. A job is said to be *small* when $p_j \leq \frac{1}{2}$ and *big* otherwise, when $p_j > \frac{1}{2}$.

The set of small jobs is denoted by J_S and the set of big jobs J_B. Note that configurations, for $T = 1$ and any machine, may contain at most one big job, while they may contain many small jobs. We denote by $B_i := \sigma^{-1}(i) \cap J_B$ the set of big jobs currently assigned by partial schedule σ to machine i, which when non-empty has big job of smallest label $\min(B_i)$. We define B_i^{min} as the singleton set containing $\min(B_i)$ when B_i is non-empty and set it to be the empty set otherwise. Note that, since σ will be a valid partial schedule during the algorithm, B_i consists always of at most 3 big jobs.

Our previous analysis would suffice if no jobs had size greater than $r = \frac{5}{6}$, which are all big jobs here. We have some slack between our dividing line of job sizes $\frac{1}{2}$ and $\frac{5}{6}$, which allows us to compensate for problems caused by jobs of size greater than $\frac{5}{6}$. We proceed to outline some of the differences between the algorithm to be introduced and that of the previous section, briefly describing the purpose of these alterations.

Changes to the algorithm are made necessary, since an invalid move (j, i) in $\mathcal{T}$ only guarantees $p(\sigma^{-1}(i)) > 1 + r - p_j$. For a big job with $p_j > r$ this may yield a value less than 1, hence insufficient to guarantee satisfaction of dual inequality constraints (5.7) when assigning dual variables as before. Two things that could be done to try and fix this problem are: (1) allow some jobs to still be allowed as *potential moves* into i, so $z(C) > y_i$ for $C \in C(i, 1)$ may indicate our algorithm is unfinished examining potential moves and (2) increase some of the y_i's up from $p(\sigma^{-1}(i))$ to exceed 1.

Addressing things in the first way leads to a notion of *undesirability* of jobs for vertices of $\mathcal{T}$ and for machines. When a vertex arising from a move (j, i) is introduced into $\mathcal{T}$, some

of the jobs are made undesirable for this vertex and set as undesirable for the machine i. Jobs still desirable (not undesirable) for i are allowed as possible potential moves, so $\sigma^{-1}(i)$ may change while (j, i) remains in $\mathcal{T}$, unlike before. The *value* of a given type of potential move (j, i) will now be made to only depend on jobs assigned to i which are undesirable for the corresponding vertex in $\mathcal{T}$, so that the value has only improved when an undesirable (for the vertex) job assigned to i is removed. This is needed for the termination argument to work as before.

We will also modify the assignment of dual variables in the analysis of the algorithm from before to accommodate the second method of trying to satisfy dual inequalities. In particular z-values will be 0 outside of a set of *active jobs* (based on the blocking tree and the notion of undesirability), and on active jobs given value $z_j = \max(p_j, r)$ corresponding to their size, possibly rounded down to r. This assignment of dual job variables essentially combines the idea of using of job sizes from earlier, together with merely counting jobs of problematic size, by having equal assignment of the dual variables for jobs of size greater than r. For each machine $i \in M$, y_i will be similar to the $z(\sigma^{-1}(i))$ from before, but here possibly adjusted up or down by $1 - r$. For machines appearing in a certain class of moves in $\mathcal{T}$, we will adjust up y_i by $1 - r$ so that we have $y_i \geq 1$. To get negative objective $\sum_{i \in M} y_i - \sum_{j \in J} z_j$ (5.5) as before, those increments must be compensated for by adjusting down y_i by $1 - r$ elsewhere at least as frequently. Machines i appearing in a move (j, i) of $\mathcal{T}$ with j small will have all jobs assigned to i active and correspond to the variables y_i we adjust down by $1 - r$. The invalidity of such (j, i) gives $p(\sigma^{-1}(i)) > 1 + r - \frac{1}{2} = \frac{8}{6}$, allowing us the room to reduce y_i while keeping it at least 1.

We proceed to describe the algorithm (and its associated definitions) in detail, which uses the ideas described above to get around the issues arising from jobs of size exceeding $r = \frac{5}{6}$.

Definition 5.14. A *blocker* is a triple (j, i, θ) with (j, i) a move and $\theta \in \{SA, BA, BL, BB\}$, such that j is small if and only if $\theta = SA$.

Blockers here play the same role in $\mathcal{T}$ that moves did previously. The special symbol in the third entry of a blocker denotes the blocker type: SA "small-to-any," BA "big-to-any," BL "big-to-least," and BB "big-to-big." The blocker types correspond to the different types of *potential moves* which our algorithm considers to grow $\mathcal{T}$. The first two entries (j, i) will as before be equal to the considered potential move. We now give a new definition for a *blocking tree* in the general case.

Definition 5.15. A *blocking tree* $\mathcal{U}$ is a pair consisting of: (1) a rooted tree on a set of vertices having a special root vertex designated as ROOT, with other vertices being blockers and (2) a list which orders the blockers that appear as vertices in $\mathcal{U}$.

We will, as before, frequently identify $\mathcal{U}$ with its corresponding list of blockers, leaving the tree structure implicit. We let $M(\mathcal{U})$ denote the set of machines appearing in the second entry among the blockers in $\mathcal{U}$.

We let $M_{SA}(\mathcal{U})$, $M_{BA}(\mathcal{U})$, $M_{BL}(\mathcal{U})$ and $M_{BB}(\mathcal{U})$ denote the sets of machines in $M(\mathcal{U})$ present in $\mathcal{U}$ with the corresponding type of blocker SA, BA, BL and BB respectively

(there may be overlap between these sets of machines). When $\mathcal{U} = \mathcal{T}$ these sets of machines are called *blocking machines* of the corresponding type.

We say a move (j, i) *appears* in $\mathcal{U}$ when some blocker of $\mathcal{U}$ has first two entries (j, i). Moves will be made to appear uniquely in $\mathcal{T}$, so we can identify blockers with their corresponding move. However, it will no longer be the case that machines can appear only once as entries to blockers of $\mathcal{T}$. It will follow from our description of the algorithm that a machine can appear in at most 3 blockers of $\mathcal{T}$, this occurring only when it is first added as a blocking machine of type BL, followed by BB and then SA.

As before we will let our blocking tree have list of blockers $\mathcal{T} := (\mathcal{B}_1, \ldots, \mathcal{B}_\ell)$ listed in the order of introduction of the blockers to $\mathcal{T}$. We let $\mathcal{T}^k := (\mathcal{B}_1, \ldots, \mathcal{B}_k)$ denote the blocking tree, on the first k blockers of $\mathcal{T}$ together with ROOT, with the same tree structure and ordering of blockers as $\mathcal{T}$.

In the definitions that follow, blocking trees $\mathcal{U}$ may be replaced with $\mathcal{T}^k$ for any k (possibly equal to $\mathcal{T}$) without causing trouble for the analysis.

Definition 5.16. Let $\mathcal{B} = (j, i, \theta)$ be a blocker. We say a job is *undesirable* for $\mathcal{B}$ according to the following list of cases by blocker-type θ:

1. If $\theta = SA$ all jobs are undesirable for $\mathcal{B}$.
2. If $\theta = BA$ all jobs are undesirable for $\mathcal{B}$.
3. If $\theta = BL$ all big jobs of label at most $\min(B_i)$ are undesirable for $\mathcal{B}$.
4. If $\theta = BB$ all big jobs are undesirable for $\mathcal{B}$.

We also have a related notion of a job being undesirable for a machine (relative to a blocking tree).

Definition 5.17. Let $\mathcal{U}$ be a blocking tree, we say a job j is *undesirable* for machine i relative to $\mathcal{U}$ if and only if there exists a blocker in $\mathcal{U}$ on machine i for which j is undesirable. Otherwise, we say j is *desirable* for machine i relative to $\mathcal{U}$.

When $\mathcal{U} = \mathcal{T}$ we simply refer to jobs as being *desirable* or *undesirable* for machines, omitting the phrase "relative to $\mathcal{U}$." Our algorithm shall want to move undesirable jobs away from blocking machines of $\mathcal{T}$ and avoid moving jobs into machines for which they are undesirable.

Definition 5.18. We associate a set of jobs $J(\mathcal{T})$ to our blocking tree $\mathcal{T}$ as follows:

$$J(\mathcal{T}) := \{j_{new}\} \cup \bigcup_{i \in M(\mathcal{T})} \{j \in J : j \text{ is undesirable for } i \text{ and } \sigma(j) = i\}.$$

$J(\mathcal{T})$ is the set of jobs which are undesirable for a machine in $M(\mathcal{T})$ they are currently assigned to by σ, together with the new job j_{new}. The set of jobs $J(\mathcal{T})$ plays the analogous role in determining potential moves as it did in the prior section.

Definition 5.19. A small job j is said to be *immovable* relative to blocking tree $\mathcal{U}$ when j is currently assigned by σ and any move (j, i) is such that j is undesirable for i relative to $\mathcal{U}$. Otherwise a small job j is said to be *movable* relative to $\mathcal{U}$.

When $\mathcal{U} = \mathcal{T}$ we simply say small jobs are movable or immovable. The set of immovable small jobs relative to $\mathcal{U}$ is denoted by $S(\mathcal{U})$, and is given explicitly by the following equation:

$$S(\mathcal{U}) = \{j \text{ small and } \sigma(j) \neq \perp : \Gamma(j) \setminus \{\sigma(j)\} \subset M_{SA}(\mathcal{U}) \cup M_{BA}(\mathcal{U})\}.$$

We define $S_i(\mathcal{U}) := S(\mathcal{U}) \cap \sigma^{-1}(i)$ to be those small jobs immovable relative to $\mathcal{U}$ that are currently assigned to machine i.

We say $S(\mathcal{T})$ is the set of immovable small jobs, and $S_i(\mathcal{T})$ is the set of immovable small jobs assigned to i. These sets of immovable small jobs shall play an important role in defining the types of potential moves.

Definition 5.20. A job $j \in J$ is said to be *active* when $j \in J(\mathcal{T})$ or $j \in S(\mathcal{T})$. The set of active jobs is denoted by $A(\mathcal{T}) := J(\mathcal{T}) \cup S(\mathcal{T})$, and $A_i(\mathcal{T}) := A(\mathcal{T}) \cap \sigma^{-1}(i)$ is defined to be the set of active jobs currently assigned to i.

Whether a job is active or not will be important in deciding how to assign its corresponding dual value in the analysis of the algorithm. We say a job j is *active for* (or activated by) machine i when $j \in A_i(\mathcal{T})$.

Definition 5.21. A *potential move* (j, i) is a move with job $j \in J(\mathcal{T})$ desirable for machine i, that does not already appear in $\mathcal{T}$, which satisfies either (a) j is small, or (b) j is big and $p(S_i(\mathcal{T})) + p_j \leq 1 + r$.

It follows from case analysis that potential moves come in one of the following mutually exclusive *types*, which will correspond to the blocker type θ of (j, i, θ), the blocker to be introduced to $\mathcal{T}$ after our algorithm considers potential move (j, i).

Type	Undesirable	Conditions	Val(j, i)
SA	All jobs	j small	$(1, \lvert\sigma^{-1}(i)\rvert)$
BA	All jobs	j big and $p(S_i(\mathcal{T}) \cup B_i) + p_j \leq 1 + r$	$(2, \lvert\sigma^{-1}(i)\rvert)$
BL	Big jobs j with $j \leq \min(B_i)$	j big with $p(S_i(\mathcal{T}) \cup B_i^{min}) + p_j > 1 + r$ and $p(S_i(\mathcal{T})) + p_j \leq 1 + r$	$(3, -\min(B_i))$
BB	Big jobs	j big with $p(S_i(\mathcal{T}) \cup B_i) + p_j > 1 + r$ and $p(S_i(\mathcal{T}) \cup B_i^{min}) + p_j \leq 1 + r$	$(4, \lvert B_i\rvert)$

The associated value Val(j, i) to potential move (j, i) is given in the final column of the above table. Potential moves are then given *priority* based on their values in lexicographically decreasing order. Note that for a potential move of type BL, the greater the minimum job, the less value and higher priority the move has. Also note that as soon as the corresponding blocker to move (j, i) is added to $\mathcal{T}$, (j, i) ceases to be potential from the condition that a potential move not appear in $\mathcal{T}$. The extra condition that (j, i) not

appear in $\mathcal{T}$ is only needed for the BL case, as in other cases the blocker (j, i, θ) introduced always has j undesirable. Finally, note that small job $j \in J(\mathcal{T})$ has a potential move (j, i) available, for some machine i, if and only if j is movable.

Updating $\mathcal{T}, \sigma$ and j_{new}: We initially set σ as the empty assignment which leaves all jobs unassigned, that is $\sigma(j) = \perp$ for all jobs j. We initially set j_{new} to be any job and $\mathcal{T}$ as the trivial blocking tree on ROOT with empty list of blockers. When there is not a valid move in $\mathcal{T}$, we extend the blocking tree by finding a potential move (j_0, i_0) of type θ which has highest priority. Let $\mathcal{B}_{add} = (j_0, i_0, \theta)$ be the corresponding blocker we wish to introduce.

1. If $j_0 = j_{new}$ we give this blocker $\mathcal{B}_{add}$ parent ROOT in the tree structure and append it to the list of blockers for $\mathcal{T}$.

2. Otherwise $i = \sigma(j_0)$ for some $i \in M(\mathcal{T})$ with j_0 undesirable for i. Let $\mathcal{B}_k$ be the first blocker in $\mathcal{T}$ with its machine $i = \sigma(j_0)$ and j_0 undesirable for i. We then give move (j_0, i_0) parent $\mathcal{B}_k$ in the tree structure and append it to the list of moves for $\mathcal{T}$.

When there is a valid move, among the moves in $\mathcal{T}$, we pick one (j_0, i_0) and update σ by setting $\sigma(j_0) \leftarrow i_0$. We update $\mathcal{T}$ by deleting some blockers as follows.

1. If $j_0 = j_{new}$ we have assigned j_{new}, so we reset $\mathcal{T}$ to the trivial blocking tree on ROOT (with empty list), and update j_{new} to some other unassigned job. If there is no unassigned job remaining the algorithm terminates, outputting σ.

2. Otherwise the blocker with move (j_0, i_0) has parent blocker $\mathcal{B}_k$ in $\mathcal{T}$. We then update $\mathcal{T}$ by $\mathcal{T} \leftarrow \mathcal{T}^{k-1}$, deleting all blockers from parent $\mathcal{B}_k$ onwards. The parent $\mathcal{B}_k$ will be such that its corresponding machine i had $i = \sigma(j_0)$ prior this update of σ.

We then describe the algorithm for producing an assignment of jobs with makespan at most $1 + r$ in pseudocode as follows:

```
1: function Build Job Assignment
2:     σ is set to the empty assignment, a constant map to ⊥
3:     while there is job unassigned by σ do
4:         j_new is set to an unassigned job
5:         T is set to trivial tree on single node ROOT, with empty blocker list
6:         while σ(j_new) = ⊥ do
7:             if there exists a move (j, i) in T which is valid then
8:                 if j = j_new then
9:                     σ(j_new) ← i
10:                else
11:                    σ(j) ← i
12:                    T ← T^{k-1} where B_k is the parent of blocker with move (j, i), deleting
    from B_k onwards in list of blockers
13:                end if
14:            else if there exists a potential move then
```

Find (j, i) a highest priority potential move and determine its type $\theta \in \{SA, BA, BL, BB\}$
Set $\mathcal{B}_{add}$ as (j, i, θ)
if $j = j_{new}$ **then**
Update $\mathcal{T}$ by adding $\mathcal{B}_{add}$ with parent ROOT and append $\mathcal{B}_{add}$ to the blocker list
else
Find first blocker $\mathcal{B}_k$ on machine $\sigma(j)$ in $\mathcal{T}$ for which j is undesirable
Update $\mathcal{T}$ by adding $\mathcal{B}_{add}$ with parent $\mathcal{B}_k$ and append $\mathcal{B}_{add}$ to the blocker list
end if
else
return "FAIL" and end function
end if
end while
end while
return σ
end function

We shall proceed to argue that this algorithm terminates and that it produces a valid partial job assignment σ with no jobs left unassigned, hence a job assignment with makespan at most $1 + r$ as desired.

Only valid moves are made to σ and the outer loop runs until we return "FAIL" or have assigned all jobs. So provided the algorithm returns σ, it produces a job assignment with makespan at most $1 + r$. So to show correctness of the algorithm we need to verify termination and that "FAIL" is never output.

Proposition 5.22. *While a blocker $\mathcal{B} = (j, i, \theta)$ continuously remains in $\mathcal{T}$ we have that $\sigma^{-1}(i)$ is unchanging (at the start of each inner loop) on jobs undesirable for this blocker.*

Proof. Suppose blocker $\mathcal{B}$ is added to $\mathcal{T}$. Subsequently considered potential moves, while $\mathcal{B}$ remains, cannot try to move an undesirable job of this blocker to i. So no undesirable jobs of $\mathcal{B}$ may be re-assigned into i by executing a valid move in $\mathcal{T}$ without deleting $\mathcal{B}$. Removing an undesirable job in this blocker from i deletes the blocker $\mathcal{B}$, since the parent of a blocker using a job currently assigned to i which is undesirable for $\mathcal{B}$ must precede (or be equal to) $\mathcal{B}$. Therefore $\sigma^{-1}(i)$ has constant intersection with the set of jobs undesirable for $\mathcal{B}$. □

Proposition 5.23. *Let $\mathcal{B}_{k+1} = (j, i, \theta)$ be a blocker added in the $k+1^{st}$ position to $\mathcal{T}$. For as long as this blocker continuously remains in $\mathcal{T}$, we have the following invariants at the start of every inner loop.*

1. *$S_i(\mathcal{T}^k)$ is invariant.*
2. *If $\theta \neq BL$ then B_i (and consequently B_i^{min}) is invariant.*

3. *If $\theta = BL$ then B_i^{min} is invariant.*

Proof. We proceed to verify each invariant.

1. For as long as the blocker $\mathcal{B}_{k+1}$ continuously remains $\mathcal{T}^k$ is constant, as otherwise this blocker would be deleted. So $M_{SA}(\mathcal{T}^k) \cup M_{BA}(\mathcal{T}^k)$ is a constant set of machines during this time. Since $M_{SA}(\mathcal{T}^k) \cup M_{BA}(\mathcal{T}^k)$ is constant, $S(\mathcal{T}^k) = \{j' \text{ small and } \sigma(j') \neq \perp : \Gamma(j') \setminus \{\sigma(j')\} \subset M_{SA}(\mathcal{T}^k) \cup M_{BA}(\mathcal{T}^k)\}$ can only change when σ either (a) changes on a job currently assigned to a machine in $M_{SA}(\mathcal{T}^k) \cup M_{BA}(\mathcal{T}^k)$ or (b) changes on a job currently in $S(\mathcal{T}^k)$. This follows, as otherwise σ can only change on small jobs by shuffling assignment among multiple machines outside $M_{SA}(\mathcal{T}^k) \cup M_{BA}(\mathcal{T}^k)$ capable of performing it. We have that (a) cannot occur by executing valid move (j', i'), as this would delete blockers starting from the SA or BA type blocker in $\mathcal{T}^k$ on machine i', hence deleting $\mathcal{B}_{k+1}$. Assuming $S(\mathcal{T}^k)$ has been constant so far after additon of $\mathcal{B}_{k+1}$, we have that $S(\mathcal{T}^k) \subset S(\mathcal{T})$ has been a set of immovable small jobs, so no subsequent potential moves have been introduced on job $j' \in S(\mathcal{T}^k)$. Therefore (b) cannot occur as σ can only change, with $\mathcal{B}_{k+1}$ remaining, by executing a potential move introduced after $\mathcal{B}_{k+1}$. Therefore $S(\mathcal{T}^k)$ is constant and σ has only been changing on jobs outside $S(\mathcal{T}^k)$. So $S_i(\mathcal{T}^k) = S(\mathcal{T}^k) \cap \sigma^{-1}(i)$ is invariant.

2. If $\theta \neq$ BL then all big jobs are undesirable for blocker $\mathcal{B}_{k+1}$. Therefore applying Proposition 5.22, we have that $B_i = J_B \cap \sigma^{-1}(i)$ is invariant, as $\sigma^{-1}(i)$ is unchanging on jobs undesirable for $\mathcal{B}_{k+1}$, including all big jobs.

3. If $\theta =$ BL then any job (of label) at most $\min(B_i)$ is undesirable for $\mathcal{B}_{k+1}$. Applying Proposition 5.22, we have $\sigma^{-1}(i)$ unchanging on those jobs, so no lesser job is assigned into machine i and the current minimum (label) job remains. Therefore $\min(B_i)$ is invariant.

So we have verified all invariants. □

Corollary 5.24. *At the start of the inner loop for current tree of blockers $\mathcal{T} = (\mathcal{B}_1, \ldots, \mathcal{B}_\ell)$ and k integer with $1 \leq k \leq \ell - 1$, we have that the following inequalities for $\mathcal{B}_{k+1} = (j, i, \theta)$ hold:*

1. *If $\theta = BA$: $p(S_i(\mathcal{T}^k) \cup B_i) + p_j \leq 1 + r$.*
2. *If $\theta = BL$: $p(S_i(\mathcal{T}^k) \cup B_i^{min}) + p_j > 1 + r$ and $p(S_i(\mathcal{T}^k)) + p_j \leq 1 + r$.*
3. *If $\theta = BB$: $p(S_i(\mathcal{T}^k) \cup B_i) + p_j > 1 + r$ and $p(S_i(\mathcal{T}^k) \cup B_i^{min}) + p_j \leq 1 + r$.*

This follows immediately from the invariants established in above proposition and conditions on potential moves ensuring that these inequalities hold to begin the iteration immediately after introduction of each blocker. □

Theorem 5.25. *The algorithm does not output "FAIL" and thus when terminating produces a job assignment of makespan at most* $1 + r$.

Proof. Suppose for contradiction that we return "FAIL" and let $j_{new}, \mathcal{T}$ and σ be the variables at the start of the inner loop iteration where this occurs. We must have no valid moves in $\mathcal{T}$ and no potential moves. We construct dual variables for the Dual CLP with $T = 1$ as follows:

$$z_j := \min(r, p_j) \qquad \forall j \in A(\mathcal{T}) \tag{5.12}$$
$$z_j := 0 \qquad \text{otherwise.} \tag{5.13}$$
$$y_i := z(A_i(\mathcal{T})) - (1 - r) \qquad \forall i \in M_{SA}(\mathcal{T}) \tag{5.14}$$
$$y_i := z(A_i(\mathcal{T})) + (1 - r) \qquad \forall i \in M_{BA}(\mathcal{T}) \tag{5.15}$$
$$y_i := z(A_i(\mathcal{T})) \qquad \text{otherwise.} \tag{5.16}$$

All active small jobs are given same value by z and p, as well as all active big jobs up to performance time r. The z-value of any job is also at most the corresponding performance time. Note also that $z(A_i(\mathcal{T})) = z(\sigma^{-1}(i))$ for each machine i, since inactive jobs are given 0 z-values.

Claim 1: (y, z) is feasible for machine scheduling Dual CLP with $T = 1$.

Recall the constraints for the machine scheduling Dual CLP (5.5)-(5.7). Clearly non-negativity constraints are satisfied, so consider constraint corresponding to machine i and $C \in C(i, 1)$ given by $y_i \geq z(C)$ (5.7). From the dual variable assignment (5.12)-(5.13) and C a configuration for target $T = 1$ we have that $z(C) \leq p(C) \leq 1$.

For any job $j \in C$ we have $z(C \setminus \{j\}) \leq p(C \setminus \{j\})$ so $z(C) \leq p(C) - p_j + z_j$. Combining with $p(C) \leq 1$ and $z_j \leq r$ this yields $z(C) \leq 1 + r - p_j$.

So combining the above bounds on $z(C)$ we observe that verifying either of the following suffices to show that the constraint $y_i \geq z(C)$ for machine i and configuration C is satisfied:

$$y_i \geq 1 \tag{5.17}$$
$$y_i \geq 1 + r - p_j \qquad \text{for some job } j \text{ in } C. \tag{5.18}$$

We have the following cases which exhaust all possibilities for i.

Case 1: $i \in M_{SA}(\mathcal{T})$.
Then there is a move (j, i) in $\mathcal{T}$ for j a small job, which is not valid. Thus $p(\sigma^{-1}(i)) + p_j > 1 + r = \frac{11}{6}$, so by $p_j \leq \frac{1}{2}$, we have $p(\sigma^{-1}(i)) > \frac{8}{6}$. Any job is undesirable for $i \in M_{SA}(\mathcal{T})$ so jobs in $\sigma^{-1}(i)$ are active. These jobs are either given same value by z and p, or are given z-value $r = \frac{5}{6}$ which is at most $1 - r = \frac{1}{6}$ less than given by p. Note if at least two jobs in $\sigma^{-1}(i)$ are given z-value r, we have $z(A_i(\mathcal{T})) \geq 2r = \frac{10}{6}$. We then conclude in any case that $z(A_i(\mathcal{T})) > \frac{8}{6} - \frac{1}{6} = 1 + (1 - r)$ and therefore $y_i = z(A_i(\mathcal{T})) - (1 - r) \geq 1$, so the constraint holds by (5.17).

Case 2: $i \in M_{BA}(\mathcal{T})$.
We must have a move (j, i) in $\mathcal{T}$ for j a big job that is not valid. So from $p_j \leq 1$ and

invalidity of move, we have $p(\sigma^{-1}(i)) > r$. Either $\sigma^{-1}(i)$ has all jobs size at most r, in which case $z(\sigma^{-1}(i)) = p(\sigma^{-1}(i))$, or a job in $\sigma^{-1}(i)$ is given z-value r. In any case we have $z(\sigma^{-1}(i) \geq r$. So from all jobs assigned to i undesirable we have $z(A_i(\mathcal{T})) \geq r$ giving result $y_i = z(A_i(\mathcal{T})) + (1 - r) \geq 1$. Thus the constraint holds by (5.17).

Case 3: $i \in M_{BB}(\mathcal{T})$.
From combining inequalities of Corollary 5.24 for a BB-blocker on machine i, we have for some k that $p(S_i(\mathcal{T}^k) \cup B_i) > p(S_i(\mathcal{T}^k) \cup B_i^{min})$. Therefore we have at least two big jobs in $\sigma^{-1}(i)$, each of which is given z-value greater than $\frac{1}{2}$, since big jobs assigned to i are active. We then have that $y_i = z(A_i(\mathcal{T})) > 1$, so the constraint holds by (5.17).

Case 4: $i \notin M(\mathcal{T})$.
Since jobs in $S_i(\mathcal{T})$ are given same p and z values and no other jobs are active for machine $i \notin M(\mathcal{T})$ we have $y_i = p(S_i(\mathcal{T}))$. If all active jobs of C are assigned by σ to machine i then $y_i = z(A_i(\mathcal{T})) \geq z(C)$ so the constraint holds. Supposing instead that there is an active job j in C not assigned to i we have that move (j, i) cannot be a potential move (this is a move by $j \in C$ implying it is assignable to i). Therefore from Definition 5.21 we have that j is big and $p(S_i(\mathcal{T})) + p_j > 1 + r$. This yields $y_i = p(S_i(\mathcal{T})) > 1 + r - p_j$ so the constraint holds by (5.18).

Case 5: $i \in M_{BL}(\mathcal{T})$ and $i \notin M(\mathcal{T}) \setminus M_{BL}(\mathcal{T})$.
As in the prior case we can assume there is an active job j in C not assigned to i. From $i \notin M_{SA}(\mathcal{T}) \cup M_{BA}(\mathcal{T})$ we have that small jobs are desirable for i. Therefore if j were small (j, i) would be a potential move a contradiction. Since configurations contain at most one big job there is a unique big job j in C and it is the only job in C not assigned to i. From $i \in M_{BL}(\mathcal{T})$ we have that $\min(B_i)$ is active, assigned to i and not in C.

Subcase 5.1 $p(\min(B_i)) \geq r$.
We have $z_{\min(B_i)} = r \geq z_j$ and other active jobs of C are assigned to i. Therefore $y_i = z(A_i(\mathcal{T})) \geq z(C)$ so the constraint holds.

Subcase 5.2 $j < \min(B_i)$.
We have $p_{\min(B_i)} \geq p_j$ and consequently $z_{\min(B_i)} \geq z_j$. Thus by other active jobs of C assigned to i, we have constraint holding $y_i = z(A_i(\mathcal{T})) \geq z(C)$.

Subcase 5.3 $p(\min(B_i)) < r$ and $j > \min(B_i)$.
Note that j is then desirable for i by $i \in M_{BL}(\mathcal{T})$, and also $z_{\min(B_i)} = p_{\min(B_i)}$. We now proceed to further subcases.

Subcase 5.3.1 (j, i) already appears as a move for a BL blocker of $\mathcal{T}$.
By Corollary 5.24 we have $p(S_i(\mathcal{T}) \cup B_i^{min}) + p_j > 1 + r$. We have that z and p agree on $S_i(\mathcal{T}) \cup B_i^{min}$ so $y_i = z(A_i(\mathcal{T})) > 1 + r - p_j$, thus the constraint holds by (5.18).

Subcase 5.3.2 (j, i) does not appear as a move for a BL blocker of $\mathcal{T}$.
By $i \notin M(\mathcal{T}) \setminus M_{BL}(\mathcal{T})$ we have that (j, i) cannot appear as a move in $\mathcal{T}$. Because (j, i) is not a potential move and j is desirable for i, we must have $p(S_i(\mathcal{T})) + p_j > 1 + r$. Therefore $y_i = z(A_i(\mathcal{T})) \geq p(S_i(\mathcal{T})) > 1 + r - p_j$, so the constraint holds by (5.18).

So in all subcases for this case we have the constraint holding.

Therefore by examination of all cases for constraints we have that (y, z) is feasible, so the first claim holds. We now check that it has negative objective.

Claim 2: (y, z) has negative objective value $\sum_{i\in M} y_i - \sum_{j\in J} z_j$ (5.5).

Note the following: $z_{j_{new}} > 0$ by the new job active, $A(\mathcal{T})$ is a disjoint union of j_{new} together with $A_i(\mathcal{T})$'s the jobs activated by each machine i and z is zero outside of the active jobs. So we can deduce the following:

$$\sum_{j\in J} z_j = z_{j_{new}} + \sum_{i\in M} z(A_i(\mathcal{T})) \tag{5.19}$$

$$\sum_{i\in M} y_i = (1-r)(|M_{BA}(\mathcal{T})| - |M_{SA}(\mathcal{T})|) + \sum_{i\in M} z(A_i(\mathcal{T})) \tag{5.20}$$

$$\sum_{i\in M} y_i - \sum_{j\in J} z_j = (1-r)(|M_{BA}(\mathcal{T})| - |M_{SA}(\mathcal{T})|) - z_{j_{new}} \tag{5.21}$$

$$\sum_{i\in M} y_i - \sum_{j\in J} z_j < (1-r)(|M_{BA}(\mathcal{T})| - |M_{SA}(\mathcal{T})|) \tag{5.22}$$

So to show the objective for this candidate solution (y, z) is negative it suffices to show that $|M_{SA}(\mathcal{T})|$ is at least $|M_{BA}(\mathcal{T})|$. Since there are no potential moves to i after the introduction of a blocker on machine i with all jobs undesirable, we have that there are no repeated machines among the blockers of type SA or BA in $\mathcal{T}$. We have no remaining potential moves and the algorithm adds SA blockers with highest priority, so it suffices to show: whenever our algorithm has a BA blocker as the final blocker in list for $\mathcal{T}$ we have a valid move in $\mathcal{T}$ or a potential move of type SA. This suffices as it would show that any blocker succeeding a BA blocker must be of type SA, and that if the final blocker was BA we would have a valid move in $\mathcal{T}$ or an SA potential move, a contradiction to the final state of $\mathcal{T}$. Thus it ensures at least as many SA blockers are in $\mathcal{T}$ as BA blockers, and therefore at least as many SA blocking machines as BA blocking machines occur in $\mathcal{T}$ by uniqueness. So we verify this sufficient condition in the following subclaim.

Sub-claim 2.1: Consider our algorithm at the start of any inner loop iteration (so σ and $\mathcal{T}$ no longer represent their final state) and suppose $\mathcal{T} = (\mathcal{B}_1, \ldots, \mathcal{B}_{k+1})$ where $\mathcal{B}_{k+1} = (j, i, BA)$. If the algorithm at this point has no valid moves in $\mathcal{T}$ then it has a potential move of type SA available.

By Corollary 5.24 we have $p(S_i(\mathcal{T}^k) \cup B_i) + p_j \leq 1 + r$, but move (j, i) is not valid so $p(\sigma^{-1}(i)) + p_j > 1 + r$. Therefore $p(\sigma^{-1}(i)) + p_j > p(S_i(\mathcal{T}^k) \cup B_i) + p_j$ implying there is some small job j_s assigned to machine i and not among the immovable small jobs $S_i(\mathcal{T}^k)$ relative to $\mathcal{T}^k$. Therefore there is a move (j_s, i') with $i' \neq i$ and $i' \notin M_{SA}(\mathcal{T}^k) \cup M_{BA}(\mathcal{T}^k) = M_{SA}(\mathcal{T}) \cup M_{BA}(\mathcal{T}) \setminus \{i\}$. Thus $j_s \in J(\mathcal{T})$ is desirable for machine i' and (j_s, i') is therefore a potential move of type SA. So the sub-claim holds.

Therefore (y, z) has negative objective value, establishing the second claim. Combining claims, we have that (y, z) is a feasible solution to the Dual CLP for $T = 1$ with negative objective value, which certifies the infeasibility of the primal, a contradiction. Therefore the algorithm does not output "FAIL." □

Theorem 5.26. *The Build Job Assignment algorithm terminates.*

Proof. As in the prior section, we associate at each stage a *signature vector* $\text{Sig}(\mathcal{T}) := (s_1, \ldots, s_\ell, \infty)$ to our blocking tree $\mathcal{T} = (\mathcal{B}_1, \ldots, \mathcal{B}_\ell)$. Here for each k, s_k is the value pair (itself ordered lexicographically) that the move corresponding to $\mathcal{B}_k$ had as a potential move just prior to its addition to $\mathcal{T}$.

The number of blockers in $\mathcal{T}$ is bounded by $|M||J|$ since every move can appear at most once in a blocker. So signature vectors have bounded length. Furthermore, any set of jobs has size at most $|J| = n$, so we have a bounded number of possible values for any potential move. Therefore we have a bounded (but exponential) number of possible signature vector values that can be taken during our algorithm.

As previously argued (see Theorem 5.9) it then suffices to show that the signature vector after adding a blocker to $\mathcal{T}$ is (strictly) less lexicographically than the signature vector after adding the previous blocker.

Let $\mathcal{T}_0 = (\mathcal{B}_1^0, \ldots, \mathcal{B}_\ell^0)$ be the blocking tree $\mathcal{T}$ after addition of the previous blocker $\mathcal{B}_\ell^0$ and let $\mathcal{T}_1 = (\mathcal{B}_1^0, \ldots, \mathcal{B}_k^0, \mathcal{B}_{add})$ for some $k \leq \ell$ be the blocking tree $\mathcal{T}$ after addition of next blocker $\mathcal{B}_{add}$. Let s_{add} be the value of the potential move corresponding to $\mathcal{B}_{add}$ just prior to addition of $\mathcal{B}_{add}$ to $\mathcal{T}$. Then we have associated signature vectors $V_0 = (s_1^0, \ldots, s_\ell^0, \infty)$ and $V_1 = (s_1^0, \ldots, s_k^0, s_{add}, \infty)$ for $\mathcal{T}_0$ and $\mathcal{T}_1$ respectively.

Case 1: $k = \ell$.
Then no valid moves were performed between addition of these blockers, and signature vectors V_0 and V_1 are identical on first ℓ entries. The $\ell + 1^{st}$ entry is ∞ for V_0 and is $s_{add} < \infty$ for V_1, therefore V_1 is less than V_0 lexicographically.

Case 2: $k < \ell$.
Let $\mathcal{B}_{k+1}^0 = (j, i, \theta)$. Let $m_f = (j_f, i_f)$ be the final valid move made between addition of blockers $\mathcal{B}_\ell^0$ and $\mathcal{B}_{add}$ to $\mathcal{T}$. Updating $\mathcal{T}$, after executing m_f, deleted blockers from $\mathcal{B}_{k+1}^0$ onwards, and returns $\mathcal{T}$ to the state it was in when (j, i) was a potential move causing introduction of $\mathcal{B}_{k+1}^0$. So $\mathcal{B}_{k+1}^0$ was the parent of blocker with move m_f, which implies that j_f in $\sigma^{-1}(i)$ was undesirable for blocker $\mathcal{B}_{k+1}^0$. By Proposition 5.22 we have that $\sigma^{-1}(i)$ was unchanging on jobs undesirable for $\mathcal{B}_{k+1}^0$ while it remained in $\mathcal{T}$. So in case all jobs are undesirable for $\mathcal{B}_{k+1}^0$ ($\theta = SA, BA$), we have that $\sigma^{-1}(i)$ was constant on all jobs for the duration of $\mathcal{B}_{k+1}^0$. Executing move m_f then reduces $|\sigma^{-1}(i)|$ from just prior to addition of $\mathcal{B}_{k+1}^0$. So (j, i) is now (after executing m_f) a potential move of the same type and with smaller value than its previous value s_{k+1}.

If $\theta = BL$, we have that $\sigma^{-1}(i)$ was constant on all big jobs (with label) at most $\min(B_i)$, and executing m_f removes $\min(B_i)$ from machine i. This either increases $\min(B_i)$ or removes the only big job from $\sigma^{-1}(i)$, so that (j, i) is now a move of lesser type BA. In any case (j, i) is now a potential move of smaller value than its previous value s_{k+1}

If $\theta = BB$, we have that $\sigma^{-1}(i)$ was constant on all big jobs and executing m_f removes a big job j_f from i, reducing $|B_i|$. So (j, i) is now a potential move of smaller value than its previous value s_{k+1} (either it is still a BB move with reduced $|B_i|$ or it is now a move with lesser type).

So for any value of θ this gives us that the highest priority move $\mathcal{B}_{add}$ has smaller value than s_{k+1}, and therefore V_1 is less than V_0 lexicographically.

Thus in any case we see a decrease in the signature vector with V_1 less than V_0. Therefore the algorithm terminates. □

So combining our results on termination and correctness of output, we have the following results.

Theorem 5.27. *Given restricted machine scheduling problem with its CLP for $T = 1$ feasible, the algorithm Build Job Assignment produces a job assignment with makespan at most $\frac{11}{6}$.* □

Theorem 5.28. *The integrality gap for the restricted machine scheduling CLP is at most $\frac{11}{6}$.* □

Theorem 5.29. *We may estimate in polytime the optimal makespan of an instance of the restricted machine scheduling problem within any factor greater than $\frac{11}{6}$.* □

Note that the algorithm given for finding the job assignment was not shown to be polytime [19]. Therefore this does not give a corresponding factor for approximating an optimal job assignment and 2 remains the best known factor of approximation for the restricted case. Improving on this factor of approximation (in either the restricted or fully general case) remains a prominent open problem in scheduling theory [19].

Chapter 6

Matroid Generalization

In a recent paper by Davies, Rothvoss and Zhang [11] a matroid generalization of the restricted Santa Claus problem was introduced and used to obtain an improved polytime approximation factor of $4+\epsilon$ for the restricted Santa Claus problem, which matches earlier work on the integrality gap of the associated CLP used for polytime estimation [4] (though not the now slightly improved factors of estimation [18, 9]).

We begin by briefly introducing the notion of a matroid. Matroids may be defined in many equivalent ways, one standard definition is as follows (see for instance a text on matroid theory by Oxley [25] as a reference).

Definition 6.1. A *matroid* is a pair $\mathcal{M} = (X, \mathcal{I})$ with finite groundset X and $\mathcal{I}$ a family of subsets of X, called the independent sets of the matroid, which satisfy the following conditions:

1. $\mathcal{I}$ is non-empty.
2. If $U \in \mathcal{I}$ then for all $V \subset U$ we have $V \in \mathcal{I}$.
3. If $U, V \in \mathcal{I}$ and $|V| < |U|$ then there exists $u \in U \setminus V$ such that $V \cup \{u\} \in \mathcal{I}$.

A *basis* of a set $A \subset X$, for a matroid with groundset X, is any maximal independent subset of A. Bases for a given set A in a matroid all have the same size, which is called the rank of A. A basis of the entire groundset X is called a basis for the matroid.

Definition 6.2. The *dual matroid* to $\mathcal{M} = (X, \mathcal{I})$ is given by $\mathcal{M}^* := (X, \mathcal{I}^*)$ with $\mathcal{I}^* := \{Y \subset X : X \setminus Y \text{ contains a basis for } \mathcal{M}\}$.

The dual matroid $\mathcal{M}^*$ to $\mathcal{M} = (X, \mathcal{I})$ is indeed itself a matroid and has bases on precisely the complements to bases of $\mathcal{M}$.

6.1 Matroid Version for Santa Claus and 4+ϵ Approximation

In the matroid setting for the restricted Santa Claus problem we have a matroid $\mathcal{M}$ defined on a groundset of players P. Instead of requiring a satisfying allocation of resources for every player, here we only require an allocation of the resources R which is satisfying for the players in some basis of the matroid $\mathcal{M}$. Formally, we have inherent resource values v_j for $j \in R$ and sets of players who can accept each resource given by $\Gamma(j)$. An allocation of resources is a collection of disjoint subsets of resources $\{S_i\}_{i \in P}$ indexed to the players where every player $i \in P$ can accept all resources in S_i. For target $T \in \mathbb{R}$, an allocation $\{S_i\}_{i \in P}$ is satisfying if and only if there exists a basis $B \subset P$ such that $v(S_i) = \sum_{j \in S_i} v_j \geq T$ for all $i \in B$. For such a basis B we say that it is satisfied by the allocation. The following LP relaxation for finding a satisfying allocation for target T and a corresponding basis of players that it satisfies was introduced in this paper [11]:

$$\sum_{j : i \in \Gamma(j)} v_j y_{ij} \geq T x_i \qquad \forall i \in P \tag{6.1}$$

$$\sum_{i \in \Gamma(j)} y_{ij} \leq 1 \qquad \forall j \in R \tag{6.2}$$

$$x \in \text{Conv}(\mathcal{B}(\mathcal{M})) \tag{6.3}$$

$$x_i \geq y_{ij} \geq 0 \qquad \forall j \in R \text{ assignable to } i \in P. \tag{6.4}$$

Here $\text{Conv}(\mathcal{B}(\mathcal{M}))$ is the convex hull of $\mathcal{B}(\mathcal{M})$, when considering bases as indicator vectors of 1's (for present players) and 0's in $\mathbb{R}^P$. A solution to this linear program with x a pure basis of players and y a $\{0, 1\}$ vector, corresponds to a satisfying allocation $\{S_i\}_{i \in P}$ for the basis x, with $S_i = \{j \in R : y_{ij} = 1\}$.

The analogous linear program relaxation for the non-matroid variant of the restricted Santa Claus problem (where each x_i is 1), is a natural relaxation to consider, by permitting fractional resource assignment. As explored by Bezáková and Dani [6], such a relaxation performs poorly when large resources (close to or exceeding the max-min) are present. By considering this matroid variant, we can circumvent this issue by allowing the matroid to capture the sets of players that can get all assigned large resources, while leaving only small resources to worry about fractionally assigning.

The following approximation result was obtained by algorithmic alternating tree techniques that extended ideas present in work due to Haxell [15] and Annamalai's work [2] that gave a constant factor (about 12.33) approximation algorithm for the restricted Santa Claus problem.

Theorem 6.3. *[11] Suppose linear program (6.2)-(6.4) is feasible for some $T > 0$. Assume that checking whether a set of players is independent in $\mathcal{M}$ can be determined in polytime. Then for any $\epsilon > 0$ there exists a polytime (in input size for the corresponding Santa Claus problem) algorithm which finds an allocation $\{S_i\}_{i \in P}$ and a basis for which it is a satisfying allocation with target $(\frac{1}{3} - \epsilon)T - \frac{1}{3} max_{j \in R} v_j$.*

The proof is rather complex and technical, so it is omitted from this discussion. We remark that the proof [11] did not involve being able to actually find a feasible solution to

the matroid relaxation (6.2)-(6.4). Instead an algorithm for finding the basis and allocation was described, which when failing was shown to certify the infeasibility of the relaxation for the target.

This result was used to obtain an improved polytime approximation factor of $4+\epsilon$ for the restricted Santa Claus problem.

Proposition 6.4. *[11] Suppose the restricted Santa Claus problem is feasible for target allocation T. Then for any $\epsilon > 0$ in polytime we may find an allocation that gives value at least $(\frac{1}{4} - \epsilon)T$ to all players.*

Proof. We partition resources into the set of *large* resources R_L with value at least $\frac{T}{4}$, and the set of small resources R_S with value less than $\frac{T}{4}$. Consider the graph G with bipartition (P, R_L) and edge set $E_G := \{\{i,j\} : j \in R_L, i \in \Gamma(j)\}$. Let $\mathcal{M} = (P, \mathcal{I})$ be the matchable set matroid of P in graph G, which has $A \subset P$ independent if and only if there exists a matching in G saturating A. Now consider the dual matroid $\mathcal{M}^* = (P, \mathcal{I}^*)$ with $\mathcal{I}^*$ consisting of the sets of players which are disjoint from some maximum matchable set in G. That is $A \subset P$ is independent in $\mathcal{M}^*$ if and only if a maximum matching for P in G has vertices contained in $(P \setminus A) \cup R_L$. Independence of $A \subset P$ in $\mathcal{M}^*$ may then be checked in polytime by using a max-matching algorithm for bipartite graphs on the induced subgraph of G with vertices $(P \setminus A) \cup R_L$ to determine whether the size of a max matching on that induced subgraph is equal to the size of a max matching on G.

Consider some (unknown) allocation $\{S_i\}_{i \in P}$ of resources that is satisfying for target T. The set of players allocated at least one large resource is matchable in G and hence contained in some basis B of $\mathcal{M}$. We have that $P \setminus B$ is a basis in the dual matroid $\mathcal{M}^*$ and that $\{S_i\}$ must allocate value at least T on small resources R_S to all elements of $P \setminus B$. Therefore the following program of form (6.2)-(6.4), using matroid $\mathcal{M}^*$, is a relaxation for the restricted Santa Claus problem and is feasible for target T (seen by taking x_i to be 1 on $P \setminus B$ and 0 on B, while taking y_{ij} to be 1 if and only if $j \in S_i$ and otherwise 0):

$$\sum_{j \in R_S : i \in \Gamma(j)} v_j y_{ij} \geq T x_i \qquad \forall i \in P \tag{6.5}$$

$$\sum_{i \in \Gamma(j)} y_{ij} \leq 1 \qquad \forall j \in R_S \tag{6.6}$$

$$x \in \mathrm{Conv}(\mathcal{B}(\mathcal{M}^*)) \tag{6.7}$$

$$x_i \geq y_{ij} \geq 0 \qquad \forall j \in R_S \text{ assignable to } i \in P. \tag{6.8}$$

Note that the maximum value of any resource in R_S is bounded by $\frac{T}{4}$. Therefore applying Proposition 6.4 to the above program (using same ϵ as given), we obtain in polytime a basis C for $\mathcal{M}^*$ and an allocation of the small resources to P, which allocates value at least $(\frac{1}{3} - \epsilon)T - \frac{1}{3}(\frac{T}{4}) = (\frac{1}{4} - \epsilon)T$ to each of the members of C. Since C is a basis for $\mathcal{M}^*$, $P \setminus C$ is a basis for $\mathcal{M}$, hence a matchable set of players in G. Thus by using a max matching algorithm on the subgraph of G induced on $(P \setminus C) \cup R_L$, we can find in polytime an assignment of large resources that gives a large resource to every player of $P \setminus C$ and therefore value at least $\frac{T}{4}$ to each of those players. Combining this assignment

of large resources with the allocation of small resources previously obtained, we produce in polytime an allocation of resources which allocates value at least $(\frac{1}{4}-\epsilon)T$ to all players. □

As an immediate consequence from our earlier remarks (see Theorem 1.6 and Proposition 1.5) on relaxed decision procedures, we have the following result for approximation.

Theorem 6.5. *For any $\epsilon > 0$, there exists a $(4+\epsilon)$-approximation algorithm for the restricted Santa Claus problem.* □

This nearly matches the best known estimation factor of $4-\frac{5}{26}$ [9] obtained from analysis of the CLP integrality gap. This is an encouraging sign that having an algorithm for the restricted machine scheduling problem which beats the long-standing factor of 2 [23], and comes close to the best known factor of estimation $\frac{11}{6}$ obtained from the CLP [19], may soon be achievable.

6.2 A Matroid Version of Machine Scheduling

A similar sort of matroid generalization may be introduced for the restricted job scheduling problem, but seems to as of yet not have been stated in the literature.

Consider a setting in which we are given set of jobs J with inherent size/performance time $p_j > 0$ for all $j \in J$, and a set M of machines. We have a matroid $\mathcal{M}$ on the groundset of jobs J and the task for a target $T > 0$ is to find a partial job assignment $\sigma : J \to M \cup \{\perp\}$ so that the set of assigned jobs (those with value not equal to the special symbol $\perp$) is a basis for $\mathcal{M}$ and no machine is assigned jobs of total size exceeding T. Recall that the machines assignable to job j are denoted by $\Gamma(j)$. A similar relaxation to that considered above is given as follows:

$$\sum_{j:i\in\Gamma(j)} p_j y_{ij} \leq T \qquad \forall i \in M \tag{6.9}$$

$$\sum_{i\in\Gamma(j)} y_{ij} \geq x_j \qquad \forall j \in J \tag{6.10}$$

$$x \in \mathrm{Conv}(\mathcal{B}(\mathcal{M})) \tag{6.11}$$

$$y_{ij} \geq 0 \qquad \forall j \in J \text{ assignable to i } \in M. \tag{6.12}$$

Note that integer solutions to this program, where x corresponds to a basis of jobs in $\mathcal{M}$ and $y_{ij} \in \{0,1\}$ for all $i \in M$ and $j \in J$, correspond to solutions for the matroid variant of the job scheduling problem with basis corresponding to x and partial assignment σ_y:

$$\begin{aligned} \sigma_y(j) &:= i \qquad \forall j \in J \text{ such that there exists (a unique) } i \in M \text{ with } y_{ij} = 1 \\ &:= \perp \qquad \text{otherwise.} \end{aligned}$$

Investigating to what extent the techniques employed for the matroid variant of the Santa Claus problem may be applicable here to obtain an approximation result is an

interesting avenue for potential research, though it is not clear how such results may help obtain corresponding results for the non-matroid variant of the problem as achieved in the Santa Claus case. It is also possible to consider similar matroid generalizations for the non-restricted cases by relaxing assumption of inherent sizes and retaining the same notion of satisfying allocations/assignments.

Chapter 7

Graph Balancing

Graph balancing problems refer to the setting of having to orient weighted edges of a graph so as to in some sense optimally balance the incoming weight to vertices. Approximation algorithms for cases related to graph balancing have given provably (provisional on $P \neq NP$) best-factor approximation results for very special cases of the Santa Claus and machine scheduling problems, which have resources/jobs assignable to at most two players/machines. This was achieved for the Santa Claus problem in 2009 with work by Chakrabarty, Chuzhoy and Khanna [7]. For the machine scheduling problem, further restrictions to the setting have been imposed to obtain this provably optimal approximation factor, as achieved in 2016 due to work by Huang and Ott [17] and independently by Page and Solis-Oba [26]. These special cases relate to graph balancing problems by noting that deciding how to assign an item among two potential recipients can be viewed as orienting an edge corresponding to the item with its endpoints the possible recipients. In the Santa Claus problem the weight for directing an edge corresponding to resource j towards player i is the valuation v_{ij}. The problem of orienting edges so as to have minimum incoming weight to any vertex maximized then corresponds to the Santa Claus problem.

We proceed to describe a part of a paper due to Chakrabarty et. al [7] which shows how we may obtain a $2 + \epsilon$ approximation algorithm (for any $\epsilon > 0$) for the case of the Santa Claus problem where resources are restricted to be assigned to at most two players, and demonstrates that a factor of 2 for approximation cannot be beaten.

7.1 Finding Balanced Orientations for Graphs with Weighted Directed Edges

We start with an exposition of the corresponding result concerning graph balancing that we apply to the Santa Claus problem. In the original paper of Chakrabarty et. al [7] the result was not proved explicitly in the form we shall give, but rather tailored specifically to the setting of the Santa Claus problem. It is hopefully instructive to see here the result first given in a graph theory setting and to see how it gives as an application the desired result for the Santa Claus problem. In the following definitions, graphs are taken to be

undirected, with multiple-edges between same vertices and loops allowed (directed edges corresponding to the graph will be considered, but not as part of its definition). We use notation $V(G)$ and $E(G)$ to denote the vertices and edges of a graph G respectively. The *degree* of a vertex is the number of undirected edges it is incident with (with loops counted only once).

Definition 7.1. Let G be a graph and $e \in E(G)$ be an edge incident to vertex $v \in V(G)$. We define e_v to be a directed edge associated to e with the same incident vertices, but directed into v.

We say e is the *undirected edge* of e_v. So each non-loop edge of G has two corresponding directed edges defined in this way, and each loop has one.

Definition 7.2. Given a graph G, we define $D(G)$, the *directed edges of* G, by

$$D(G) := \{e_v : e \in E(G) \text{ and } v \text{ is incident with } e\}.$$

We may also refer to the directed edges as *arcs*, but note that since G is a multi-graph, directed edges are not uniquely determined by an ordered pair of vertices. For each vertex $v \in V(G)$ we define $\text{in}_G(v)$ and $\text{out}_G(v)$ to be the set of directed edges in $D(G)$ directed into and out of v respectively. Note that we consider a loop on vertex $v \in V(G)$ to be both directed into and out of v.

Definition 7.3. A *(non-negative) weight function* w on directed edges of a graph G is a function $w : D(G) \to \mathbb{R}^+$. Additionally, for each vertex $v \in V(G)$, let $w^G_{max}(v) := \max(\{w(d) : d \in \text{in}_G(v)\} \cup \{0\})$ be the maximum weight of any edge directed into v, or 0 if there are no such directed edges.

Definition 7.4. An *orientation of edges* for a graph G, is a function $\mathcal{O} : E(G) \to D(G)$ such that $\mathcal{O}(e) = e_v$ for some vertex $v \in V(G)$ incident to e, for all edges $e \in E(G)$.

We say that $\mathcal{O}(e)$ is the *orientation of edge* e, the choice of directed edge made by $\mathcal{O}$ on edge e. We also use $\mathcal{O}$ to denote the range of the function, the set of directed edges it chooses.

Definition 7.5. Suppose G is a graph with orientation of edges $\mathcal{O}$ and weight function on directed edges w. Let $v \in V(G)$. The *weight oriented into* v *by* $\mathcal{O}$ is defined as $\sum_{d \in \text{in}_G(v) \cap \mathcal{O}} w(d)$.

Theorem 7.6. *Let G be a graph with a weight function on its directed edges w. Let $M : V(G) \to \mathbb{R}$ be a real valued function on vertices with $M(v)$ denoted by M_v for all $v \in V(G)$. Suppose we have the total weight of all directed edges into each vertex v satisfying:*

$$\sum_{d \in D(G) : d \in in_G(v)} w(d) \geq M_v + w^G_{max}(v) \qquad \forall v \in V(G). \tag{7.1}$$

Then there exists an orientation of edges $\mathcal{O}$ such that the weight oriented into any vertex $v \in V(G)$ by $\mathcal{O}$ is at least $\frac{M_v}{2}$, and moreover such an orientation may be found in polytime.

Proof. Given input (G, w, M) we say $\mathcal{O}$ is a *desired orientation* if it is an orientation of edges for G which orients weight at least $\frac{M_v}{2}$ into each vertex $v \in V(G)$. To show an algorithm $\mathcal{A}$ finds such an orientation in polytime, it is enough to show that $\mathcal{A}$, when given input (G, w, M) as above, finds a desired orientation and that the algorithm runs in polytime when given access to $\mathcal{A}$ running on a graph with total number of vertices and edges less than $|E(G)| + |V(G)|$ as a single operation oracle. Such an algorithm will be constructed in the following induction argument, and it will be clear from its recursive construction that it possesses the desired properties.

We proceed by induction on $|V(G)| + |E(G)|$ to show that given (G, w, M) satisfying conditions of the theorem, there is an algorithm $\mathcal{A}$ that orients weight at least $\frac{M_v}{2}$ into each vertex v. Note that in base case that G is the trivial empty graph with no vertices the result holds with the empty orientation, produced by a trivial algorithm $\mathcal{A}_0$.

Let $n \in \mathbb{Z}^+$ and suppose for any natural $k < n$ there exists an algorithm $\mathcal{A}_k$, which given input satisfying conditions of the theorem on a graph with total number of vertices and edges equal to k, produces a desired orientation. We wish to produce an algorithm $\mathcal{A}_n$, which takes as input (G, w, M) satisfying the conditions of the theorem with graph G having $n = |V(G)| + |E(G)|$, and produces a desired orientation.

So let G be a graph having $|V(G)| + |E(G)| = n$ total number of vertices and edges, with weight function w and values associated to vertices M, satisfying condition (7.1). G a graph with at least one vertex, so we can find by exhaustive search in polytime a vertex $u \in V(G)$ of minimum degree. We then consider the following cases, which cover all possibilities:

Case 1: Vertex u has degree 0.
We have $w^G_{max}(u) = 0$ and $\sum_{d \in D(G): d \in \text{in}_G(u)} w(d) = 0$, so for the condition (7.1) to hold we must have $M_u \leq 0$. Note that the conditions of the theorem are also met for subgraph $H := G \setminus u$, with same weight function w and using same values M_v. So by hypothesis $\mathcal{A}_{n-1}$ finds an orientation of edges for H given by $\mathcal{O}$ such that for all $v \in V(G) \setminus \{u\}$ the weight oriented into v by $\mathcal{O}$ is at least $\frac{M_v}{2}$. Since $E(G) = E(H)$ and $M_u \leq 0$ we have that $\mathcal{O}$ is an orientation for G which orients into each vertex $v \in V(G)$ at least weight $\frac{M_v}{2}$. We set $\mathcal{A}_n$ to output this same $\mathcal{O}$.

Case 2: Vertex u has degree 1.
We have $\sum_{d \in D(G): d \in \text{in}_G(u)} w(d) = w^G_{max}(u)$, so for the condition (7.1) to hold we must have $M_u \leq 0$. Let $e \in E(G)$ be the unique edge incident to u.

If e is a loop, then as before conditions of the theorem are met by the subgraph $H := G \setminus u$ with same (restricted) weight function w and values M_v. Thus running $\mathcal{A}_{n-2}$ on input H produces an orientation for H which orients weight at least $\frac{M_v}{2}$ into each vertex v of H. This, together with orienting loop e in only possible way, provides an orientation for G which orients at least weight $\frac{M_v}{2}$ into each vertex $v \in V(G)$ as desired. We set $\mathcal{A}_n$ to output this resulting orientation.

Suppose e is not a loop and let z be the other vertex incident to e. Consider the graph H formed from G by identifying vertex u with z, so e is now considered a loop on the vertex z. H has vertices $V(G) \setminus \{u\}$ and edges $E(G)$, where edge e is in H identifying a loop on z and

all other edges have endpoints as before in G. From this identification we have that w gives directed loop edge e_z in $D(H)$ same weight as w gives the directed edge e_z from u to z in $D(G)$. For all $v \in V(G) \setminus \{u\}$ we have that $\sum_{d \in D(G):d \in \text{in}_G(v)} w(d) = \sum_{d \in D(H):d \in \text{in}_H(v)} w(d)$ and $w^G_{max}(v) = w^H_{max}(v)$, in particular this holds for z because of our choice of weight on loop e_z to match the previous weight from u to z. So with same weight function w and values M_v we have H satisfying the conditions (7.1). Thus applying the inductive hypothesis we have that $\mathcal{A}_{n-1}$ produces an orientation $\mathcal{O}$ for the edges of H which orients at least weight $\frac{M_v}{2}$ into each vertex $v \in V(G) \setminus \{u\}$. From the identification of edge e in G between u and z with the loop on z in H, and the corresponding identification of directed edge e_z, we have that $\mathcal{O}$ is an orientation for the edges of G. This orientation $\mathcal{O}$ for G orients weight at least $\frac{M_v}{2}$ into each vertex $v \in V(G) \setminus \{u\}$ and $0 \geq \frac{M_u}{2}$ into u. So $\mathcal{O}$ is a desired orientation of edges for G and we set it as the output for $\mathcal{A}_n$.

Case 3: G has minimum degree at least 2.
Examining the set of directed edges of $D(G)$, for each $v \in V(G)$ we can find in polytime $d_{v,1}$ and $d_{v,2}$ the directed edges of $\text{in}_G(v)$ which have highest and second-highest weights by w respectively (breaking ties arbitrarily). Consider the directed graph $\mathcal{D}$ on $V(G)$ with set of arcs $\bigcup_{v \in V(G)} \{d_{v,1}, d_{v,2}\}$. Note that $\mathcal{D}$ has in-degree 2 for all vertices by definition. If $\mathcal{D}$ is loopless, then we greedily try to construct a long directed path in $\mathcal{D}$ by choosing a new start vertex from an in-neighbour to the previous start vertex to add to the path at each stage. When all in-neighbours to the current start vertex s have been previously added to the path, we pick a choice for directed edge $d \in \{d_{s,1}, d_{s,2}\}$ so that its undirected edge (job) is not the same as that of the first directed edge (out from s) of our current path. Adding d to the path creates a unique directed cycle with all underlying edges unique.

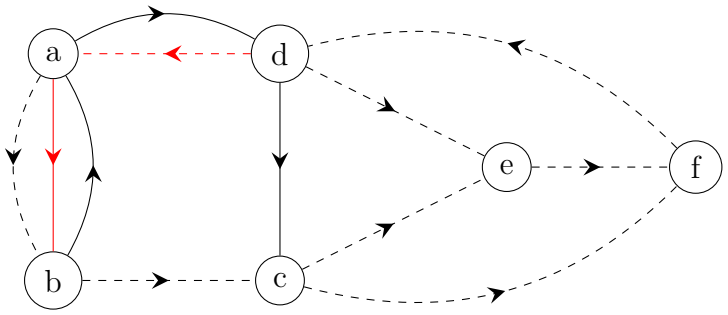

Figure 7.1: (Theorem 7.6) An example directed graph $\mathcal{D}$ is shown with the path constructed (together with final arc) drawn with solid lines and other directed edges of $\mathcal{D}$ with dashed lines. Different colours (red and black) are used to distinguish different underlying edges (jobs) to directed edges between the same vertices. This path was constructed starting from c then expanded with new start vertices d, a, b respectively. Subsequently the path could no longer be extended as the only in-neighbour to b is a, which previously appeared. So a final directed edge, differing on underlying edge from that already present between a and b, is added to form a directed cycle C on vertices $\{a, b\}$.

In any case, taking a loop or such a directed cycle described above, we can in polytime find a directed cycle C in $\mathcal{D}$ which uses at most one directed edge for every (undirected) edge of G. We define a graph $H := G \setminus C$ from G and C, by deleting all undirected edges

of C from G and retaining the same vertex set $V(H) = V(G)$. We consider the same (restricted) weight function w on the directed edges $D(H)$ of H. Let $v \in V(H)$.

1. If v is not a vertex of C, then v has the same weights of edges directed into it in $D(H)$ as in $D(G)$. Therefore:
$$\sum_{d \in D(H): d \in \text{in}_H(v)} w(d) = \sum_{d \in D(G): d \in \text{in}_G(v)} w(d) \geq M_v + w^G_{max}(v) = M_v + w^H_{max}(v). \quad (7.2)$$

2. If v is a vertex of C, we have a unique directed edge into v from the directed cycle C, let this be c_v. By definition $w(d_{v,2}) \leq w(c_v) \leq w(d_{v,1}) = w^G_{max}(v)$. Suppose the undirected edge to $d_{v,1}$ is not present in C, then the directed edges from C into v have weight at most $d_{v,2}$ (so deleting C removes at most weight $2w(d_{v,2})$ on directed edges into v) and $w^H_{max}(v) = w^G_{max}(v)$. So, recalling inequality from condition (7.1), we have the following:
$$\begin{aligned} \sum_{d \in D(H): d \in \text{in}_H(v)} w(d) &\geq \sum_{d \in D(G): d \in \text{in}_G(v)} w(d) - 2w(d_{v,2}) && (7.3)\\ &\geq M_v + w^G_{max}(v) - 2w(d_{v,2}) && (7.4)\\ &\geq (M_v - 2w(c_v)) + w^H_{max}(v). && (7.5) \end{aligned}$$
Supposing instead that $d_{v,1}$ is present in C, then we have removed weight at most $w(d_{v,1}) + w(d_{v,2})$ on directed edges into v and $w^H_{max}(v) \leq w(d_{v,2})$. So we have a similar calculation, yielding the same inequality:
$$\begin{aligned} \sum_{d \in D(H): d \in \text{in}_H(v)} w(d) &\geq \sum_{d \in D(G): d \in \text{in}_G(v)} w(d) - w(d_{v,1}) - w(d_{v,2}) && (7.6)\\ &\geq M_v - w(d_{v,2}) && (7.7)\\ &\geq (M_v - 2w(c_v)) + w^H_{max}(v). && (7.8) \end{aligned}$$

Let $k = |V(H)| + |E(H)| < n$. We run algorithm $\mathcal{A}_k$ on H, with same (restricted) weight function w and vertex values determined from the above calculations, that is vertex values M_v for v outside C (7.2) and $M_v - 2w(c_v)$ for a vertex of C (7.5). By the inductive hypothesis this must produce an orientation $\mathcal{O}'$ for H which satisfies the following:

1. If v is not a vertex of C then the weight oriented into v by $\mathcal{O}'$ is at least $\frac{M_v}{2}$.
2. If v is a vertex of C then the weight oriented into v by $\mathcal{O}'$ is at least $\frac{M_v}{2} - w(c_v)$, where c_v is the directed edge from C into vertex v.

We construct $\mathcal{O}$ an orientation of edges for G from $\mathcal{O}'$ and C as follows. We make the same choice of orientation for edges in $E(H)$ as in $\mathcal{O}'$ and we use the directed cycle C to choose orientation for the edges appearing in the corresponding undirected cycle to C. Since C does not use the same undirected edge twice, this produces a well-defined orientation.

Let $v \in V(G)$.

1. If v is not incident to directed cycle C, then it follows that $\mathcal{O}$ orients at least weight $\frac{M_v}{2}$ into v, the same as $\mathcal{O}'$.

2. If v is a vertex of C then we have that $\mathcal{O}'$ orients weight at least $\frac{M_v}{2} - w(c_v)$ into v. Now $\mathcal{O}$ retains that orientation from $\mathcal{O}'$ and also orients (the corresponding undirected edge of) c_v into v. So $\mathcal{O}$ orients weight at least $(\frac{M_v}{2} - w(c_v)) + w(c_v) = \frac{M_v}{2}$ into v.

Therefore we have obtained a desired orientation of edges $\mathcal{O}$ which we set as the output of $\mathcal{A}_n$.

This covers all cases, so by induction we have algorithms A_n for each $n \in \mathbb{N}$ as desired. This shows the existence of an orientation of edges, for any (G, w, M) satisfying conditions of the theorem, which orients at least $\frac{M_v}{2}$ into each vertex $v \in V(G)$. Let $\mathcal{A}$ be the algorithm corresponding to $\{\mathcal{A}_n : n \in \mathbb{N}\}$ the algorithms implicitly constructed in the induction proof, which runs on input (G, w, M) by finding $n = |V(G)| + |E(G)|$ and gives the same output as $\mathcal{A}_n$ (on same input). We note from our inductive construction that $\mathcal{A}$ runs by calling itself on input with lesser $|V(G)| + |E(G)|$ and performs other operations taking polytime. Therefore $\mathcal{A}$ is a polytime algorithm for finding desired orientations. □

7.2 Optimal Factor Approximation for Santa Claus Graph Balancing

Now we return to the Santa Claus problem with each resource assignable to at most two players. For each resource $j \in R$ we let a_j and b_j be the players j may be assigned to, where $a_j = b_j$ if and only if j is assignable to just one player. We do not confine ourselves to the restricted case here, so we may have $v_{a_j,j} \neq v_{b_j,j}$. We say such instances of the Santa Claus problem are in the *directed graph balancing* (or *unrelated graph balancing*) setting and in the *graph balancing* setting when resources have inherent values $v_j = v_{a_j,j} = v_{b_j,j}$. This same terminology is also used for the machine scheduling problem, when replacing values with job performance times.

Definition 7.7. Let graph $G_0 := (P, R)$ have vertex-set equal to set of players P and have edge-set R corresponding to resources, where resource $j \in R$ has incident vertices a_j and b_j.

Note that this is an undirected graph, potentially having loops and multi-edges. We have resource j assignable to player i if and only if edge j is incident to vertex i in G_0.

We define a weight function w on directed edges of G_0 by giving weight v_{ij} to a directed edge into player i corresponding to resource j. So the weight of an edge/resource directed into a vertex/player, is the value of the resource to that player.

Let T be a target allocation. Consider the associated equality version of the CLP relaxation for the problem (3.11)-(3.14). Recall that $C(i, T)$ is the set of all bundles of

resources assignable to player i with total value at least T.

Santa Claus Graph Balancing Equality CLP

$$\sum_{S \in C(i,T)} x_{i,S} = 1 \qquad \forall i \in P \tag{7.9}$$

$$\sum_{i \in \{a_j, b_j\},\ S \in C(i,T):\ j \in S} x_{i,S} = 1 \qquad \forall j \in R \tag{7.10}$$

$$x_{i,S} \geq 0 \qquad \forall i \in P \text{ and } S \in C(i,T)). \tag{7.11}$$

The above is a relaxation for the Santa Claus problem and is therefore feasible for $T = 0$ (when everyone is satisfied to receive nothing). Moreover, in polytime we may find to any desired accuracy the optimal T for which this is feasible and a corresponding feasible solution. So to have a $2 + \epsilon$ approximation algorithm for this class of instances, it suffices (see Theorem 3.7) to in polytime produce an allocation giving value at least $\frac{T}{2}$ to all players when given a feasible solution of the above linear program for target T.

Suppose linear program (7.9)-(7.11) is feasible for some T and let $x = (x_{i,S})$ be a feasible solution.

Definition 7.8. Let $j \in R$ and $i \in P$, we define $y_{ij} := \sum_{S \in C(i,T):\ j \in S} x_{i,S}$ to be the weight with which x assigns resource j to player i.

Note that $y_{ij} = 0$ for any $i \notin \{a_j, b_j\}$. Also notice that inequality (7.10) can be rewritten to give the following equality:

$$\sum_{i \in P} y_{ij} = 1 \qquad \forall j \in R. \tag{7.12}$$

Definition 7.9. We say resource $j \in R$ is *integrally allocated* to player i (by x) when $y_{ij} = 1$.

Definition 7.10. We say a resource is *integrally allocated* when it is integrally allocated to some player and otherwise we say the resource is *fractionally allocated*.

Note that all resources having exactly one player that it can be assigned to are integrally allocated. Also note that when a resource j is integrally allocated to some player then by inequality (7.12) we have $y_{ij} = 0$ for all other players. For each player i let I_i be the set of resources integrally allocated to i, and let $I = \bigcup I_i$ be the set of all integrally allocated resources. Let $T_i := T - \sum_{j \in I_i} v_{ij}$ be the target remaining for each player i to receive on fractionally allocated resources given that we allocate to i each of the resources it is integrally allocated by x.

Definition 7.11. The graph $G_1 = (P, R \setminus I)$ is obtained from G_0 by deleting all the integrally allocated resources.

We consider w as a weight function on directed edges of G_1 by taking its restriction on $D(G_1)$. To produce an allocation of resources satisfying target $\frac{T}{2}$ for each player it suffices to produce an orientation of edges $\mathcal{O}$ for G_1 so that the weight oriented into each vertex $i \in P$ by $\mathcal{O}$ is at least $\frac{T_i}{2}$. Here the orientation of edges corresponds to an assignment of the fractionally allocated resources, which we combine with assigning integrally allocated resources to the corresponding player they are integrally allocated to. This suffices since the weight oriented into each player $i \in P$ by $\mathcal{O}$ makes up at least half the difference between T and the value that was already integrally allocated to i by x.

Proposition 7.12. *If $i \in P$ then the total weight on directed edges $D(G_1)$ into i is at least $T_i + w_{max}^{G_1}(i)$.*

Proof. Let $i \in P$. Suppose first that i is a degree 0 vertex of G_1, so $w_{max}^{G_1}(i) = 0$. Then any resource j with $y_{ij} > 0$ is integrally allocated to i. Let $S \in C(i, T)$ be some bundle of resources allocated by x to player i with $x_{i,S}$ greater than 0. All the resources of configuration S must therefore be integrally allocated to i, so $T_i = T - \sum_{j \in I_i} v_{ij} \leq 0$. Therefore the total weight on directed edges of G_1 into i is $0 \geq T_i + w_{max}^{G_1}(i)$.

Now suppose that i is not isolated, so $w_{max}^{G_1}(i) = w(d) = v_{ij_0}$ for a directed edge $d \in D(G_1)$ corresponding to some fractionally allocated resource j_0 directed into player i. Since j_0 is fractionally allocated and equation (7.9) holds, we have some bundle of resources $S \in C(i, T)$ not containing j_0 with $x_{i,S} > 0$. Since S is a configuration for player i with target T, we must have the total value to player i of its resources at least T. So the total value to player i of fractionally allocated resources in S is at least $T - \sum_{j \in I_i} v_{ij} = T_i$. The total weight on directed edges $D(G_1)$ into i is the total value given by player i to the fractionally allocated resources assignable to i, hence at least $T_i + v_{ij_0} = T_i + w_{max}^{G_1}(i)$.

So in any case we have the desired result holding. □

From Theorem 7.6 we can in polytime find an orientation $\mathcal{O}$ for G_1, such that the weight oriented into each player $i \in P$ is at least $\frac{T_i}{2}$. Combined with our earlier observation, this shows we can in polytime find an allocation for this instance of the Santa Claus problem satisfying target $\frac{T}{2}$ (an integral solution to the CLP for $\frac{T}{2}$) as desired.

Theorem 7.13. *Suppose we have an instance of the Santa Claus problem with all resources assignable to at most two players. In polytime a solution to the equality CLP (7.9)-(7.11) for target T may be rounded to an integral solution for target $\frac{T}{2}$.* □

Theorem 7.14. *For any $\epsilon > 0$ there is a $2 + \epsilon$ approximation algorithm for the class of instances of Santa Claus problem with all resources assignable to at most two players.* □

We also have a 2-hardness result for estimation of max-min in this setting, for even the (restricted case) graph balancing setting, based on reduction from a NP-complete 3CNF variant of the 3SAT problem for Boolean satisfiability as shown by Chakrabarty et. al [7].

We will use the following terminology from propositional logic. A *Boolean variable* is a variable taking possible values TRUE or FALSE. A *literal* is a Boolean variable x or its

negation $\neg x$. Given a formula in propositional logic of form $(\ell_{1,1} \vee \ldots \vee \ell_{1,n_1}) \wedge \ldots \wedge (\ell_{m,1} \vee \ldots \vee \ell_{m,n_m})$ where $\ell_{a,b}$ are literals, then each bracketed portion $(\ell_{a,1} \vee \ldots \vee \ell_{a,n_a})$ is a *clause*. Note a clause is made TRUE by an assignment of Boolean variables if and only if (at least) one of its literals is made TRUE, and that the entire formula is made TRUE if and only if all clauses are made TRUE.

Definition 7.15. A formula ϕ in propositional logic is *satisfiable* when there exists an assignment of TRUE or FALSE to the Boolean variables of ϕ which makes the formula TRUE.

Definition 7.16. A *3CNF formula* is a formula in propositional logic of form $(\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1,3}) \wedge \ldots \wedge (\ell_{m,1} \vee \ell_{m,2} \vee \ell_{m,3})$ with exactly 3 literals in each clause.

Here we insist on exactly 3 literals, frequently this condition is stated as there being at most 3. We now introduce a variant of the satisfiability problem for 3CNF formulas.

Definition 7.17. The *3CNF Boolean satisfiability problem* (or 3SAT) is the task of determining whether a given 3CNF formula ϕ with each literal appearing in at most 2 clauses is satisfiable.

The 3CNF problem was one of the first problems known to be NP-complete (along with other variants of Boolean satisfiability) due to pioneering work of Cook and Levin [10, 24]. The minimal structure of the 3CNF problem makes it adaptable to deriving other NP-hardness results by showing that determining satisfiability of a 3CNF formula can be reduced to solving some other problem [3].

Given an instance of the 3CNF problem with formula ϕ we define an associated instance of the restricted Santa Claus problem in the graph balancing setting $\mathcal{S}(\phi)$ as follows:

1. Let the set of players P be given by the clauses and literals appearing in ϕ, we call these *clause players* and *literal players* respectively.

2. For each variable x appearing in ϕ we have a *variable resource* corresponding to x with value 2 assignable to exactly those literals corresponding to x in ϕ (any of x or $\neg x$ in ϕ).

3. For each clause C appearing in ϕ we have a *clause resource* corresponding to C with value 1 and the resource is assignable only to player C.

4. For each pair (ℓ, C) with C a clause of ϕ and ℓ a literal appearing in C we have a *pair resource* (ℓ, C) assignable to exactly players ℓ and C. The value of this pair resource will be 2 if the literal ℓ appears in only one clause, and 1 if it appears in two clauses.

Proposition 7.18. *[7] Let ϕ be a 3CNF formula from an instance of the 3CNF problem. The associated instance of Santa Claus problem $\mathcal{S}(\phi)$ has max-min at least 2 if and only if ϕ is satisfiable.*

Proof. Suppose that ϕ is satisfiable. Then there exists an assignment of truth values to variables of ϕ such that each clause of ϕ has at least one TRUE literal, and we pick such a variable assignment f. We assign clause resources to the corresponding clause. We assign variable resource x to the corresponding literal, x or $\neg x$, of ϕ made TRUE by f, if no such literal appears in ϕ we leave x unassigned (or assign it arbitrarily). For each pair resource $j = (\ell, C)$ with ℓ TRUE under f, assign j to clause C. Pair resources with ℓ FALSE are assigned to literal ℓ.

We check that the resulting allocation from this assignment of resources, allocates at least value 2 to each player.

1. If player C is a clause, then it is allocated the corresponding clause resource and also at least one pair resource (ℓ, C) with ℓ TRUE under f. This allocates value at least 2 to C.

2. If player ℓ is a literal which is TRUE under f, then ℓ is allocated a variable resource of value 2.

3. If player ℓ is a literal which is FALSE under f, then ℓ is assigned all pair resources containing literal ℓ. This either allocates one resource of value 2 to ℓ or two of value 1.

So indeed this allocation gives at least value 2 to all players, so the max-min for $\mathcal{S}(\phi)$ is at least 2.

Now suppose that the max-min of $\mathcal{S}(\phi)$ is at least 2 and we consider some allocation $\{S_i\}_{i \in P}$ giving value at least 2 to all players. Suppose x is a variable resource which is allocated to one of its literals, then we assign the variable x so that this literal is made TRUE. This is possible as allocations consist of disjoint bundles, which implies variable resources are allocated to at most one literal player. With other Boolean variables (left unassigned by the allocation) we assign truth values arbitrarily. Call the resulting variable assignment g. Let C be any clause appearing in ϕ. Since C is allocated value at least 2, it must be allocated some pair resource (ℓ, C). Now ℓ is also allocated value at least 2, and as ℓ is not allocated all of its possible pair resources, it must have been allocated its corresponding variable resource. Therefore ℓ is made TRUE by g, so the clause C is also made TRUE. Therefore the entire formula ϕ is made TRUE by g. So ϕ is satisfiable.

Therefore the max-min of $\mathcal{S}(\phi)$ is at least 2 if and only if ϕ is satisfiable. □

Since the max-min of $\mathcal{S}(\phi)$ is integer, when less than 2 the max-min is at most 1. This shows that solving the 3CNF problem for formula ϕ is equivalent to estimating the max-min of the associated instance of the Santa Claus problem to strictly within a factor of 2.

Theorem 7.19. *For the Santa Claus problem in the graph balancing setting, it is NP-hard to estimate the max-min strictly within a factor of* 2. □

This shows that the $2+\epsilon$-approximation result proved above is essentially the best possible. So in the graph balancing and directed graph balancing settings, the problem of producing approximately optimal solutions to the Santa Claus problem has been nicely resolved. In fact further work by Verschae and Wiese [29] has been done to get a 2-approximation algorithm for this same setting, that avoids using rounding from the CLP (which introduces an ϵ difference to the factor of approximation).

7.3 Graph Balancing for Machine Scheduling

In the analogous class of instances for machine scheduling related to graph balancing, those instances with jobs having inherent size and being performable on at most two machines, we do not yet have a provably optimal $\frac{3}{2}$ approximation algorithm. However, in a special subclass of these instances a provably optimal approximation factor of $\frac{3}{2}$ has been obtained by Huang and Ott [17] and independently by Page and Solis-Oba [26]. This optimal case for restricted machine scheduling has that in addition to a restriction on the number of machines capable of doing each job, there are only two job sizes allowed. The setting in which this optimal makespan approximation result is obtained, is in another respect slightly more general than the graph balancing setting, as only the jobs of the larger size need be restricted to having at most two machines capable of performing them [17]. With just the restriction to two performance times (no longer in graph balancing setting at all), work by Chakrabarty, Khanna and Li [8] has also beaten the general 2-approximation mark.

The general graph balancing and directed graph balancing settings for machine scheduling still have unresolved polytime approximation factors and are the subject of some recent research, including an improvement in the estimation factor for graph balancing, beating the long-standing factor of $\frac{7}{4}$. This improvement in estimation comes out of a 2018 paper due to Jansen and Rohwedder [20], which slightly improves upon the $\frac{7}{4}$ factor from 2008 work due to Ebenlendr, Krčál and Sgall [12]. The improved factor of estimation arises from bounding the integrality gap for the associated CLP, but does not yield the corresponding approximation factor. The $\frac{7}{4}$ factor does also apply to approximation, as we will see from exploring the algorithm that obtains it [12]. Note that $\frac{7}{4}$ compares favourably to the best known factor for the general restricted case estimation of $\frac{11}{6}$ discussed earlier. These results for graph balancing have been obtained by using the machine scheduling CLP [20], or some (as we shall argue) weaker relaxation [12]. Interestingly then, in the directed graph balancing case for machine scheduling, where jobs take different amounts of time to perform by the two players, it is known that the CLP has an integrality gap of 2 [29]. This makes the directed graph balancing case in some sense fundamentally harder to approach than its graph balancing counterpart or even the general restricted case.

We proceed to discuss the algorithm by Ebenlendr, Krčál and Sgall [12] which achieves the $\frac{7}{4}+\epsilon$ approximation ratio for machine scheduling graph balancing. Their paper uses a less restrictive relaxation than the CLP [12], but we shall present it here using the CLP we have already discussed and shall show that it suffices.

Once again graphs will have loops and multiple-edges allowed, and we will recall Definition 7.2 for directed edges and Definition 7.4 for orientations of the edges of graphs.

Definition 7.20. Let $G = (V, E)$ be a graph. A *fractional orientation of edges* for G, is a function $f : D(G) \to [0, 1]$ such that for all edges $e \in E$, the sum of f over all (one or two) corresponding directed edges to e is equal to 1.

By interpreting $f(d) = 1$ as assigning edge e to d, for e the undirected edge for directed edge d, a fractional orientation of edges with integer values corresponds to a (pure) orientation of edges (see Definition 7.4). Given a graph $G = (V, E)$ and a positive weight function on edges $w : E \to (0, \infty)$, for f a fractional orientation of edges, we say that f has *load* $\sum_{d \in in_G(v)} w(d)f(d)$ on each vertex $v \in V$. The *makespan* for a fractional orientation of edges f is the maximum load on any vertex in graph G.

Consider an instance of the machine scheduling problem in the graph balancing case with machines M, jobs J having sizes p_j for all $j \in J$ and with job j assignable to machines $\{a_j, b_j\}$ (possibly not distinct). We define a corresponding graph $G = (M, J)$ with edge $j \in J$ incident to vertices a_j and b_j, and a weight function on edges given by the job sizes p_j. Given a directed edge d corresponding to job j, we will also write $p_d := p_j$. Job assignments with makespan at most T correspond exactly to the orientations of the edges of G with makespan at most T.

It suffices to show (by a machine scheduling version of Theorem 3.7) that given a solution x to the machine scheduling equality CLP for target T, we can produce a job assignment with makespan at most $\frac{7}{4}T$. The equality version of the CLP (with objective omitted) for this class of instances is given as follows:

$$\sum_{S \in C(i,T)} x_{i,S} = 1 \qquad \forall i \in P \tag{7.13}$$

$$\sum_{i \in \{a_j, b_j\},\ S \in C(i,T):\ j \in S} x_{i,S} = 1 \qquad \forall j \in R \tag{7.14}$$

$$x_{i,S} \geq 0 \qquad \forall i \in P \text{ and } S \in C(i, T)). \tag{7.15}$$

By scaling we may suppose we are given x a solution to (7.13)-(7.15) for target $T = 1$. This implies all jobs have size at most 1. We apply Definition 7.8 to the machine scheduling CLP with solution x, where we define $y_{ij} := \sum_{S \in C(i,1), j \in S} x_{i,S}$ to be the weight with which x assigns job j to machine i.

Note that x determines a fractional orientation of edges f for G as follows: for directed edge d of job j into machine $i \in \{a_j, b_j\}$ we set $f(d) := y_{ij}$. Equality (7.14) ensures that this satisfies the requirement of having the sum of f over possible orientations for j equal to 1. The makespan for f is also at most 1, from equality (7.13) and configurations having at most total job size 1. We seek to transform fractional orientation f into a pure orientation of edges with makespan at most $\frac{7}{4}$, yielding a desired job assignment. We will update a fractional orientation g, initialized as f, to progress towards getting a pure orientation of the edges.

We say an edge of G is *integrally oriented* (by g) when one of its corresponding directed edges d has $g(d) = 1$, and otherwise the edge is fractionally oriented. Note that loop edges are always integrally oriented.

We distinguish between *big* jobs, with $p_j > \frac{1}{2}$ and call other jobs *small*, and make the corresponding distinction between *big edges* and *small edges*. We define H^g to be the spanning subgraph of G taking only the fractionally oriented edges. When this graph H^g is empty, then g corresponds to a (pure) orientation of edges in G. We define H^g_B to be the spanning subgraph of H^g taking only the big edges.

One of the crucial operations [12] to perform on g, will be to take a cycle of H^g and circulate load on the cycle, until some job/edge in the cycle is now integrally oriented.

Rotate(H^g, C): For H^g as above and C a directed cycle of H^g we do the following:
1. Find $m = \min_{d \in C} p_d(1 - g(j))$.
2. For all directed edges d in C increase $g(d)$, with update $g(d) \leftarrow g(d) + \frac{m}{p_d}$. Also reduce g on the reverse arc to d by the same amount $\frac{m}{p_d}$ so that g remains a fractional orientation.

This rotation operation pushes load m around the cycle, not affecting the load on any vertex. By definition of m, g after the update will have values on C staying in range $[0, 1]$, thus staying a fractional orientation. Also there exists (at least one) directed edge d of C, such that applying the rotation sets $g(d) = 1$, thus integrally orienting the corresponding edge. So H^g after performing Rotate(H^g, C) has a reduced number of edges.

The other operation to define, will be on a rooted tree, where we integrally orient all edges of the tree downwards from the root.

PushDown($H^g, \mathcal{T}$): Let $\mathcal{T}$ be a rooted tree in H^g. For every directed edge d outwards (or downwards) from root, we set $g(d) = 1$ and $g(d') = 0$ for d' the reverse arc inwards/up to the root.

Suppose every edge in the tree $\mathcal{T}$ is small. We have that g orients edges downwards from a root, thus it orients at most one of these (small) edges into each vertex v (and orients others away) and therefore increases load by at most $\frac{1}{2}$ on any vertex.

Note that these two operations defined on H^g may be performed in polytime. If no big edges/jobs were present in the initial fractional graph H^f, we could simply do the following to obtain a $\frac{3}{2}$ approximation algorithm (matching the earlier work for such a case [23]): update g by repeated rotation of cycles until H^g is a forest, then pick roots for the resulting trees and orient them downwards.

To get around this issue of big jobs, we will show that f coming from the CLP ensures some additional properties [12] on the fractional orientation, that we will maintain for g throughout the algorithm. This was not shown in the paper giving this $\frac{7}{4}$-approximation result [12], where these additional properties to impose on fractional orientations were introduced as linear constraints, so we demonstrate this explicitly here.

Definition 7.21. For tree $\mathcal{T}$ we define $L(\mathcal{T})$ to be the set of directed edges into the leaf (degree 1) vertices of $\mathcal{T}$.

Then we produce this result (remarked to be true [20] by Jansen and Rohwedder) that having f coming from the CLP guarantees one of the needed properties for the initial fractional orientation.

Proposition 7.22. *Let $\mathcal{T}$ be a tree, which is a subgraph of initial $H_B^g = H_B^f$, then $\sum_{d\in L(\mathcal{T})} p_d f(d) \geq \sum_{d\in L(\mathcal{T})} p_d - 1$.*

Proof. The result is trivial for a 2 vertex tree. In such a case with unique edge e in $\mathcal{T}$ with directed edges d_1 and d_2, we must have $f(d_1) + f(d_2) = 1$ and $1 \geq p_e > \frac{1}{2}$. Thus $p_e f(d_1) + p_e f(d_2) > 1 \geq 2p_e - 1 = p_{d_1} + p_{d_2} - 1$ as desired.

Let $\mathcal{T}$ be a tree on at least 3 vertices (so no edge goes between leaf vertices) that is a subgraph of H_B^f. Any bundle of jobs that is a configuration for any player with target 1 may contain at most one big job.

From the CLP equalities (7.13)-(7.15) we have that $|V(\mathcal{T})| = \sum_{i\in V(\mathcal{T}),S\in C(i,1)} x_{i,S}$. Let $\mathcal{T}'$ be the tree formed from $\mathcal{T}$ with leaf edges removed.

Since jobs from $E(\mathcal{T})$ are in unique bundles, this also implies that $|V(\mathcal{T}')|$ is at least the total weight over all jobs in $E(\mathcal{T})$ that get assigned by x into $V(\mathcal{T}')$. Recalling definition for y_{ij}'s this yields:

$$|V(\mathcal{T}')| \geq \sum_{i\in V(\mathcal{T}'),j\in E(\mathcal{T})} y_{ij} \tag{7.16}$$

$$= \sum_{i\in V(\mathcal{T}'),j\in E(\mathcal{T}')} y_{ij} + \sum_{i\in V(\mathcal{T}'),j\in E(\mathcal{T})\setminus E(\mathcal{T}')} y_{ij}. \tag{7.17}$$

Since jobs of $E(\mathcal{T}')$ can only be accepted by its two incident vertices in the tree $E(\mathcal{T}')$, by equality (7.14), we have:

$$\sum_{i\in V(\mathcal{T}'),j\in E(\mathcal{T}')} y_{ij} = |E(\mathcal{T}')| = |V(\mathcal{T}')| - 1. \tag{7.18}$$

The edges of $E(\mathcal{T})\setminus E(\mathcal{T}')$ are those corresponding to directed edges in $L(\mathcal{T})$. For $d \in L(\mathcal{T})$ with undirected edge e, CLP solution x gives weight $f(d)$ to bundles allocating job/edge e to a leaf vertex of $\mathcal{T}$. Therefore a weight $1 - f(d)$ is on bundles allocating e to a vertex of $\mathcal{T}'$, so:

$$\sum_{i\in V(\mathcal{T}'),j\in E(\mathcal{T})\setminus E(\mathcal{T}')} y_{ij} = \sum_{d\in L(\mathcal{T})} (1 - f(d)) \tag{7.19}$$

$$= |L(\mathcal{T})| - \sum_{d\in L(\mathcal{T})} f(d). \tag{7.20}$$

Combining (7.16, 7.17, 7.18, 7.20) yields inequality:

$$|V(\mathcal{T}')| \geq (|V(\mathcal{T}')| - 1) + (|L(\mathcal{T})| - \sum_{d\in L(\mathcal{T})} f(d)). \tag{7.21}$$

Rearranging we obtain $\sum_{d\in L(\mathcal{T})} f(d) \geq |L(\mathcal{T})| - 1$. Using this fact in the following inequalities we obtain:

$$\sum_{d\in L(\mathcal{T})} p_d f(d) = \sum_{d\in L(\mathcal{T})} p_d - \sum_{d\in L(\mathcal{T})} p_d(1 - f(d)) \tag{7.22}$$

$$\geq \sum_{d\in L(\mathcal{T})} p_d - \sum_{d\in L(\mathcal{T})} (1 - f(d)) \tag{7.23}$$

$$= \sum_{d\in L(\mathcal{T})} p_d - |L(\mathcal{T})| + \sum_{d\in L(\mathcal{T})} f(d) \tag{7.24}$$

$$\geq \sum_{d\in L(\mathcal{T})} p_d - |L(\mathcal{T})| + (|L(\mathcal{T})| - 1) \tag{7.25}$$

$$= \sum_{d\in L(\mathcal{T})} p_d - 1. \tag{7.26}$$

Thus we obtain the desired inequality. □

Proposition 7.23. *Initially $H_B^g = H_B^f$ is a disjoint union of trees and cycles.*

Proof. Let X be a component of H_B^f. If X does not contain a cycle, it is a tree as desired. So suppose that X has a cycle C. As noted previously, no two big jobs/edges of $E(C)$ can be contained in any configuration for the CLP. So from CLP equality (7.14) summed over jobs of $E(C)$ we have $|E(C)| = \sum_{i\in V(C), j\in E(C)} y_{ij}$. Since C is a cycle $|E(C)| = |V(C)|$ and from CLP equality 7.13 we have total weight given by x on bundles to a machine in $V(C)$ equal to $|V(C)|$. Thus no bundle S with $x_{i,S} > 0$ for $i \in V(C)$ exists without containing exactly one job from $E(C)$. Since any bundle containing a big job from $E(X \setminus C)$ cannot contain a big job from $E(C)$, supposing $f(d) = \sum_{S\in C(i,1): j\in S} x_{i,S} > 0$ for some directed edge d from a vertex of $V(X) \setminus V(C)$ to vertex i in C would be a contradiction. Thus C is a component of X and therefore all of X. So components of H_B^f are trees or cycles as desired. □

We define our algorithm for updating g as follows [12]. When considering a particular edge e between vertices u and v it will be convenient to write $e = \{u, v\}$ with its directed edges written as $e_v = uv$ and $e_u = vu$. When we use this notation there will only be a single edge between u and v, so no ambiguity is introduced.

1: **function** Graph Balancing Rounding
2: Initialize $g \leftarrow f$.
3: **while** H^g is non-empty graph **do**
4: **if** There exists a vertex of degree 1 in H^g **then**
5: Find one one such vertex v and find incident edge $e = \{u, v\}$ for some vertex u of H^g.
6: **if** $p_e g(vu) \geq \frac{3}{4}$ **then**
7: Find the maximal tree in H_B^g containing big edge e and root it at u to form rooted tree $\mathcal{T}$.
8: Update g by applying PushDown(H^g,$\mathcal{T}$).

else
Set $g(uv) \leftarrow 1$ and $g(vu) \leftarrow 0$.
end if
else
Start at some vertex of non-zero degree and add edges of H^g to build a (non-backtracking) walk outwards until a directed cycle C is formed as part of the walk. When possible choose the new edge to be a big edge.
Update g by applying Rotate(H^g,C).
end if
end while
return g
end function

Note that the only operations performed on g are PushDown, Rotate and the taking of one edge of degree 1 and reassigning it integrally. Each of these preserves g as a fractional orientation, and H^g being empty means that all edges of G are integrally oriented by g. So when terminating and returning g, an orientation of edges for G is produced.

Note that the algorithm does not get stuck by making an incorrect assumption on the current state. Indeed $e = uv$ being such that $p_e g(vu) \geq \frac{3}{4}$ for fractional orientation g implies $p_e \geq \frac{3}{4}$ is big. Also H^g having no vertex of degree 1 ensures that a directed cycle C will be found in polytime by the described method.

All steps run in polytime and after each iteration of the loop, additional edges become integrally oriented by g. Once an edge becomes integrally oriented by g, it remains integrally oriented throughout. Therefore the Graph Balancing Rounding runs in polytime.

H^g_B is a subgraph of H^f_B at all points, so it remains a disjoint union of trees and cycles throughout.

Proposition 7.24. *[12] Throughout the algorithm g retains the property that for any tree $\mathcal{T}$ a subgraph of H^g_B we have $\sum_{d\in L(\mathcal{T})} p_d f(d) \geq \sum_{d\in L(\mathcal{T})} p_d - 1$.*

Proof. It holds initially from Proposition 7.22. Note that PushDown can only occur in the algorithm on a rooted tree that is an entire component of H^g_B, since the tree is chosen maximally and degree 1 vertices in H^g_B can only occur on tree components. Therefore PushDown deletes a component of H^g_B, thus retaining this property.

When single leaf edges are integrally oriented, this either has no effect on H^g_B or deletes one edge, leaving g with same values on other directed edges of H^g_B. Thus this property is retained on all the remaining trees of H^g_B.

So it remains to verify that property $\sum_{d\in L(\mathcal{T})} p_d f(d) \geq \sum_{d\in L(\mathcal{T})} p_d - 1$ for any tree $\mathcal{T}$ of H^g, is not lost by applying Rotate(H^g,C) on some directed cycle C chosen by our algorithm. If C has an edge coming from a cycle component X of H^g_B, then it follows from prescribed choices of edges in the walk that the constructed C is equal to the cycle X in some orientation. So after applying this rotation, the cycle component X has become a disjoint union of paths, where any sub-path has one directed edge into a leaf aligned with

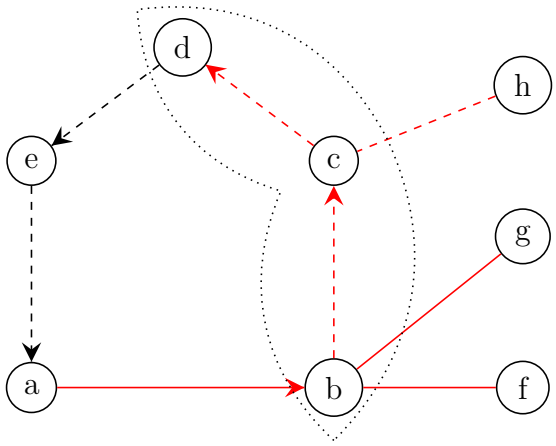

Figure 7.2: (Proposition 7.24) A directed cycle C is shown together with a component of H_B^g it intersects. Big edges are in red, others in black. The tree $\mathcal{T}$ is shown with solid lines, other edges are dashed. $\mathcal{T}$ is extended to $\mathcal{T}'$ by extending a single path, out from non-leaf endpoint b of the intersecting path of $\mathcal{T}$ and C, to include as many consecutive big edges of the cycle as possible. This extension is shown in a dotted border.

C and one opposed. Hence the sum of $p_d g(d)$ over directed edges into leaf vertices remains unchanged for any tree of H_B^g with vertices in $V(X)$.

Consider a tree $\mathcal{T}$ contained in a tree component Y of H_B^g and let C be a directed cycle chosen to rotate by our algorithm. The intersection of the edges in C with $\mathcal{T}$ must be a disjoint union of paths. Let $\mathcal{P}$ be such a path with exactly one endpoint a leaf vertex of $\mathcal{T}$ and the other end a non-leaf vertex u such that the directed edge into u in $\mathcal{T}$ is aligned with C. We extend the path $\mathcal{P}$ on one end beyond u maximally such that the resulting path is contained in C with all edges big. This must extend $\mathcal{P}$ by at least one edge, since otherwise the walk generating C should have chosen another big edge from $\mathcal{T}$ incident to u, when extending out from u. Replace each such path $\mathcal{P}$ in $\mathcal{T}$ with its extension (see Figure 7.2), to obtain a larger tree $\mathcal{T}'$, which is a subtree of Y. Pushing load m through cycle C affects the load that $L(T') \cap E(C)$ edges put on leaf vertices by $\pm m$ depending on the alignment with C. By extending those intersecting paths $\mathcal{P}$, we ensured that at least as many directed edges of $L(\mathcal{T}')$ are aligned with C as reverse aligned. So in total $\sum_{d \in L(\mathcal{T}')} p_d g(d)$ does not decrease and therefore $\sum_{d \in L(\mathcal{T}')} p_d g(d) \geq \sum_{d \in L(\mathcal{T}')} p_d - 1$ is preserved. Note that $L(\mathcal{T}) \subset L(\mathcal{T}')$ as $\mathcal{T}'$ extends $\mathcal{T}$ only out from non-leaf vertices of $\mathcal{T}$, and therefore by subtracting off all p_d for $d \in L(\mathcal{T}') \setminus L(\mathcal{T})$, we obtain (using $g(d) \leq 1$) result $\sum_{d \in L(\mathcal{T})} p_d g(d) \geq \sum_{d \in L(\mathcal{T})} p_d - 1$ as desired.

Therefore all performed operations on g preserve this inequality for all trees $\mathcal{T}$ of H_B^g. □

Theorem 7.25. *The makespan of the g returned by Graph Balancing Rounding is at most $\frac{7}{4}$.*

Proof. Applying Rotate does not effect the load on any vertex. Consider application of PushDown to a maximal tree $\mathcal{M}$ in H_B^g with leaf vertex root v and edge $e = \{u, v\}$ such

that $p_e g(vu) = p_e(1 - g(uv)) > \frac{3}{4}$. So by Proposition 7.24, for any subtree $\mathcal{T}$ of $\mathcal{M}$ rooted at v, we have $\sum_{d \in L(\mathcal{T}) \setminus \{uv\}} p_d g(d) \geq \sum_{d \in L(\mathcal{T}) \setminus \{uv\}} p_d - \frac{1}{4}$. So for any leaf vertex $w \neq v$ of subtree $\mathcal{T}$, with directed edge d into w, we have $p_d(1 - g(d)) < \frac{1}{4}$. Therefore PushDown integrally orienting d to w, increases the load by less than $\frac{1}{4}$. Since every vertex of $\mathcal{M}$ may be viewed as a leaf to a subtree of $\mathcal{M}$ (by deleting its children), this shows that PushDown increases the load on any one vertex by at most $\frac{1}{4}$. After completion of PushDown, the component $\mathcal{M}$ of H_B^g is deleted, and therefore PushDown is never applied twice on a tree containing a given vertex.

In the remaining way that g updates, the directed edge uv is made to have $g(uv) = 1$, such that previously $p_{uv} g(vu) \leq \frac{3}{4}$. So $g(uv)$ increases by at most $\frac{3}{4}$ and hence the load on v increases by at most $\frac{3}{4}$. Our algorithm chooses v to have degree 1 in H^g, therefore after this operation, v is isolated in H^g and v's load is fixed. If PushDown has not previously been applied to a tree containing v, then v has increased from its original load by at most $\frac{3}{4}$.

Note that if PushDown has previously been used on a tree (of big edges) containing v, then prior to integrally orienting $\{u, v\}$, we have that v was isolated in H_B^g. Thus edge $\{u, v\}$ is small and v's load therefore increased by at most $\frac{1}{2}$ from orienting $\{u, v\}$. Combined with earlier observation that the load increases on a vertex by less than $\frac{1}{4}$ from applying PushDown, we also conclude that the load has increased by at most $\frac{3}{4}$ throughout the algorithm.

Originally $g = f$ was a fractional orientation for G with makespan at most 1, so the g returned as output, is a (pure) orientation with makespan at most $1 + \frac{3}{4} = \frac{7}{4}$. □

So by scaling of the problem for arbitrary target T this algorithm shows the following results:

Theorem 7.26. *Suppose we have an instance of the graph balancing version of the machine scheduling problem. In polytime a solution to the equality CLP (7.13)-(7.15) for target T may be rounded to an integral solution for target $\frac{7}{4}T$.* □

Theorem 7.27. *For any $\epsilon > 0$ there is a $\frac{7}{4}$-approximation algorithm for the machine scheduling problem in the graph balancing case.* □

As discussed above, this work [12] gives the best known result for approximation in the graph balancing case and near the best estimation result [20]. The algorithm exploits the division between big jobs and small jobs, and the known integrality gap of 2 [29] for the CLP in the directed graph balancing case indicates that the approach taken here is fundamentally not generalizable to this case.

The Rotate algorithm however, can be adjusted for the directed graph balancing case, as we shall demonstrate. Given a cycle to rotate, we will choose to orient the cycle so that load can be pushed through along this orientation, without causing an increase in load to any vertex of the cycle.

Rotate2(G,C,w,h): Given loopless multigraph G, cycle C of G, weight function on directed edges w, and a fractional orientation h, which does not integrally orient any edge

of C, we do the following:
1. Find the two directed cycles for C: $C_1 = (c_1, \ldots, c_n)$ and reverse oriented $C_2 = (c'_n, \ldots, c'_1)$ with each c'_i the reverse arc of c_i.
2. If $\frac{\prod_{k=1}^{n} w(c_k)}{\prod_{k=1}^{n} w(c'_k)} \leq 1$ pick orientation $D := C_1$ with directed edges $(d_1, \ldots, d_n) = (c_1, \ldots, c_n)$. Otherwise, pick $D := C_2$ with directed edges $(d_1, \ldots, d_n) = (c'_n, \ldots, c'_1)$. Let d'_i be the reverse arc to each d_i.
3. Define a list α by $\alpha_1 = 1$ and $\alpha_k = \frac{\prod_{i=1}^{k-1} w(d_i)}{\prod_{i=2}^{k} w(d'_i)}$ for all $k \in \{2, \ldots, n\}$.
4. Find $m = \min_{k \in \{1,\ldots,n\}} \frac{1}{\alpha_k}(1 - h(d_k))$.
5. Let $h(d_k) \leftarrow h(d_k) + m\alpha_k$ and decrease on reverse arc d'_k by setting $h(d'_k) \leftarrow h(d'_k) - m\alpha_k$ for all $k \in \{1, \ldots, n\}$.

Note that α_k was chosen so that for all $k \geq 2$ the increase in load coming into a vertex from d_{k-1} is balanced by the decrease in load coming from d'_k. By the choice of orientation $\frac{\prod_{k=1}^{n} w(d_k)}{\prod_{k=1}^{n} w(d'_k)} \leq 1$, and this will ensure that increase in load coming into the starting vertex from d_n is at least balanced out by the decrease on d'_1.

So this rotation algorithm does not increase the load on any vertex and adjusts h to integrally orient some edge of the cycle. Combining this with orienting outwards from trees yields the standard [23] 2-approximation factor when starting from any fractional orientation meeting the target makespan.

It remains an open problem whether a factor better than 2 can be achieved for estimation or approximation in the directed graph balancing case of the machine scheduling problem.

Chapter 8

Bipartite Hypergraph Covering

For $\mathcal{H} = (A \cup X, \mathcal{E})$ a bipartite hypergraph (see Definition 4.3) we say a set of edges that hits each vertex in A at most once is an *A-side matching*. We will again use notation $\mathcal{E}_a$ to denote the set of edges of $\mathcal{E}$ which use A-vertex $a \in A$. As seen in Chapter 4, the machine scheduling problem may be framed as the task of finding an A-side matching which covers all of X, whereas for the Santa Claus problem the related task on hypergraphs was to find an A-saturating matching. Haxell's Theorem [15] (Theorem 4.8) gives a condition, in terms of the maximum edge size, on all subsets of A which guarantees an A-saturating matching. Here we introduce and investigate what is a similar sort of condition on subsets of X and what this guarantees on the portion of X that can be covered in some A-side matching. We will also explore how the conditions introduced correspond to the CLP for machine scheduling (5.1)-(5.4).

8.1 Conditions for Covering

First we investigate how the type of condition on transversals used in Haxell's Theorem [15] arises from the dual problem to max matchings, and in a similar way we shall find a type of condition coming from a dual problem to trying to find A-side matchings covering X.

An LP formulation for a max matching in a hypergraph $\mathcal{H} = (A \cup X, \mathcal{E})$ is as follows:

$$\max \quad \sum_{E \in \mathcal{E}} x_E \tag{8.1}$$

$$\sum_{E \in \mathcal{E}: v \in E} x_E \leq 1 \qquad \forall v \in A \cup X \tag{8.2}$$

$$x_E \geq 0 \qquad \forall E \in \mathcal{E}. \tag{8.3}$$

Here feasible solutions $x \in \{0,1\}^{\mathcal{E}}$ with objective value m correspond exactly to the matchings in $\mathcal{H}$ of size m (hence saturating m vertices of A). So verifying that the objective

value for feasible solutions is bounded by some amount less than $|A|$, verifies that no A-saturating matching is possible. Such bounds may be obtained by weak duality applied to the following dual program:

$$\min \sum_{v \in A \cup X} y_v \tag{8.4}$$

$$\sum_{v \in E} y_v \geq 1 \qquad \forall E \in \mathcal{E} \tag{8.5}$$

$$y_v \geq 0 \qquad \forall v \in A \cup X. \tag{8.6}$$

Any dual objective value to a feasible solution bounds the objective for the primal matching LP. Consider a candidate solution $y \in \{0,1\}^{A \cup X}$ to the dual (8.4)-(8.6) which is zero on exactly $B \subset A$ vertices of A. Automatically y satisfies constraints for edges hitting A outside of B, and for other edges with $E \cap B \neq \emptyset$ constraint 8.5 says that y satisfies the corresponding constraint if and only if y is 1 for some X vertex of E. This is equivalent to saying that y is a feasible dual solution if and only if $Y := \{x \in X : y_x = 1\}$ is a transversal for $\mathcal{H}_B$ (see definition 4.6). Note that the objective for candidate solution y is equal to $|Y| + (|A| - |B|)$ and recall that objective values less than $|A|$ of feasible solutions to the dual verify that no A-saturating matching exists in $\mathcal{H}$.

So suppose there exists $B \subset A$ such that $\mathcal{H}_B$ has a transversal $U \subset X$ of size less than $|B|$, then we have a corresponding feasible dual solution y obtained by setting y_a for $a \in A$ zero on B and 1 otherwise in A and setting y_x for $x \in X$ to be 1 on U and zero otherwise. This has objective $(|A| - |B|) + |U| < |A|$ thus certifying that no A-saturating matching exists in $\mathcal{H}$. So a necessary condition for an A-saturating matching that arises from this associated linear program is that for any $B \subset A$ we have any transversal of $\mathcal{H}_B$ with size at least $|B|$. This can also be seen to be a necessary condition, by noting that for any matching $\mathcal{M}$ saturating $B \subset A$, a transversal of $\mathcal{H}_B$ must hit X-vertices in each of the $|B|$ edges from $\mathcal{M}$ intersecting B, and from disjointness of edges these are all unique, hence any transversal has size at least $|B|$. Haxell's Theorem 4.8 says that by strengthening this necessary condition in a manner depending of the maximum edge size we obtain a sufficient condition for having an A-saturating matching. The sufficient condition from Haxell's Theorem [15] is that, for r the size of the maximum edge in $\mathcal{H}$, transversals of $\mathcal{H}_B$ for any $B \subset A$ have size greater than $(2r-3)(|B|-1)$, which essentially strengthens the necessary condition inequality between transversal sizes of $\mathcal{H}_B$ and $|B|$ by a factor of $2r-3$.

Now we present a natural LP formulation for an A-side matching which covers X and find a similar sort of necessary condition for having such a cover, which we shall consider

attempting to strengthen into potentially a sufficient condition.

$$\max \quad 0 \tag{8.7}$$

$$\sum_{E\in\mathcal{E}:v\in E} x_E \geq 1 \qquad \forall v \in X \tag{8.8}$$

$$\sum_{E\in\mathcal{E}_a} x_E \leq 1 \qquad \forall a \in A \tag{8.9}$$

$$x_E \geq 0 \qquad \forall E \in \mathcal{E}. \tag{8.10}$$

Here feasible solutions $x \in \{0,1\}^{\mathcal{E}}$ correspond exactly to the A-side matching covers for X. The dual program, which we use to obtain a necessary condition for feasibility of primal (8.7)-(8.10) is given below.

$$\min \quad \sum_{a\in A} y_a - \sum_{v\in X} z_v \tag{8.11}$$

$$y_a \geq \sum_{v\in E\setminus\{a\}} z_v \qquad \forall a \in A, E \in \mathcal{E}_a \tag{8.12}$$

$$y_a, z_v \geq 0 \qquad \forall a \in A, v \in X. \tag{8.13}$$

Note that this primal dual pair corresponds closely to the pair of LPs seen for machine scheduling (5.1)-(5.7). Indeed the machine scheduling CLP is a special case of the above formulation with edges in the hypergraph arising from configurations. For an instance of the machine scheduling problem on machines M and jobs J with target T, by taking $A = M$, $B = J$ and $\mathcal{E} = \bigcup_{i\in M}\{\{i\} \cup S : S \in C(i,T)\}$ we have that primal-dual pairs (5.1)-(5.7) and (8.7)-(8.13) are identical.

So as discussed for the CLP, from linear programming duality and fact that the dual (8.11)-(8.13) is feasible with all zeros, we have that the primal relaxation for A-side matchings covering X (8.7)-(8.10) is feasible if and only if the optimal/minimum objective for the dual is 0.

Consider a candidate solution $(y,z) \in \mathbb{Z}^A_{\geq 0} \times \{0,1\}^X$ to the dual program (5.1)-(5.7) which takes value 1 on $Y \subset X$ and value 0 for X vertices outside Y. For any edge E hitting $a \in A$, the corresponding constraint (8.12) is satisfied if and only if $y_a \geq |E \cap Y|$. So (y,z) is feasible if and only if for all $a \in A$ we have $y_a \geq \max_{E\in\mathcal{E}_a}|E \cap Y|$.

Definition 8.1. For bipartite hypergraph $\mathcal{H} = (A \cup X, \mathcal{E})$ we define a function $f : \{0,1\}^X \to \mathbb{Z}_{\geq 0}$ on subsets of X by $f(Y) := \sum_{a\in A} \max_{E\in\mathcal{E}_a}|E \cap Y|$ for all $Y \subset X$.

Given $Y \subset X$, we have a feasible solution $(y,z) \in \mathbb{Z}^A_{\geq 0} \times \{0,1\}^X$ given by taking z_v for $v \in X$ to be 1 on Y and zero otherwise on X, with y_a set to be $\max_{E\in\mathcal{E}_a}|E \cap Y|$ for all $a \in A$. This feasible solution has objective value $f(Y) - |Y|$ which is less than 0 if and only if $f(Y) < |Y|$.

Therefore a necessary condition for the feasibility of the primal (8.7)-(8.10), and hence the existence of an A-side matching covering X, is the following:

$$f(Y) \geq |Y| \qquad \forall Y \subset X. \tag{8.14}$$

The above shows how the condition can be seen to arise from associated linear programs, but the condition is also easily seen to be necessary by noting that $f(Y)$ bounds how many vertices of Y counted with multiplicity can appear among the edges of an A-side matching.

As before we can consider a similar strengthening of this condition on bipartite hypergraph $\mathcal{H} = (A \cup X, \mathcal{E})$ by some factor which may depend on the maximum edge size of our hypergraph. Firstly we consider a strengthening (*) of this necessary condition by some absolute constant $c \geq 1$ independent of edge size:

$$f(Y) \geq c|Y| \qquad \forall Y \subset X. \quad (*) \tag{8.15}$$

We shall show that this condition guarantees the existence of an A-side matching which covers some constant fraction (depending on c, but at least $\frac{1}{3}$) of the vertices in X, but in general cannot guarantee the existence of an A-side matching which covers X.

We shall generalize this condition (*) somewhat to situations with weighted X-vertices.

Definition 8.2. Let $w : X \to \mathbb{R}_{\geq 0}$ be a non-negative weight function on X. We set $w(X') := \sum_{x \in X'} w(x)$ on subsets $X' \subset X$ accordingly, and define a function $f_w : 2^X \to \mathbb{R}$ by $f_w(Y) := \sum_{a \in A} \max_{E \in \mathcal{E}_a} w(Y \cap E)$ for all $Y \subset X$.

The weighted analogue of condition (*) with constant c and weighting w is as follows:

$$f_w(Y) \geq cw(Y) \qquad \forall Y \subset X. \tag{8.16}$$

Suppose this condition 8.16 holds with constant $c = 1$ for all non-negative weightings and let (y, z) be a feasible solution for the dual (8.11)-(8.13). Note that z is a non-negative weighting and that $y_a \geq z(E \setminus \{a\})$ for all $E \in E_a$. Therefore the objective value is at least $f_z(X) - z(X) \geq 0$. So the dual has optimal objective 0 and therefore the primal (8.7)-(8.10) is feasible.

Conversely, suppose the weighted condition 8.16 is violated for weight function w with $c = 1$ on the set $Y \subset X$. Define $z_v := w(v)$ for all $v \in Y$ and $z_v = 0$ for $v \in X \setminus Y$, and set $y_a := \max_{E \in \mathcal{E}_a} w(E \cap Y)$. Note that $w(S \cap Y) = z(S)$ for any $S \subset X$. Thus $y_a \geq z(E \setminus \{a\})$ for all $a \in A$ and $E \in \mathcal{E}_a$, giving feasibility for (y, z) in dual (8.11)-(8.13). The objective value of this solution is $\sum_{a \in A} y_a - \sum_{v \in X} z_v = f_w(Y) - w(Y)$, which is less than 0 since we supposed Y to be violating 8.16 for w with $c = 1$. This certifies the infeasibility of the primal (8.7)-(8.10).

Therefore feasibility of the relaxation for A-side matchings covering X (8.7)-(8.10) is equivalent to condition 8.16 holding for $c = 1$ with any non-negative weighting. In earlier chapters we have reviewed some of the implications (for obtaining integral solutions) of feasibility for the machine scheduling (5.1)-(5.4), which is an LP we have seen to be a special case of (8.7)-(8.10). So a discussion of the sorts of conditions on bipartite hypergraphs introduced here is also a natural generalization of such work.

8.2 A Greedy Algorithm for Covering and Some Negative Results

Let $\mathcal{H} = (A \cup X, \mathcal{E})$ with w a non-negative weight function on X and set $m := |A|$. We proceed to describe a "greedy" method for constructing an A-side matching which covers a large portion of the weight of X, provided conditions of form equation 8.16 are met.

function GREEDY A-SIDE MATCHING
  Initialize variables $B \leftarrow A$, $Y \leftarrow X$ and $\mathcal{U} \leftarrow \emptyset$
  Initialize index $i \leftarrow 1$
  while $i \leq m$ **do**
    Find an edge E_i with $w(E_i \cap Y)$ max such that $E_i \cap B \neq \emptyset$
    $B \leftarrow B \setminus E_i$
    $Y \leftarrow Y \setminus E_i$
    $\mathcal{U} \leftarrow \mathcal{U} \cup \{E_i\}$
    $i \leftarrow i + 1$
  end while
  return $\mathcal{U}$
end function

This algorithm returns the set of edges $\mathcal{U}$, which was built up in stages by adding edges which covered as much weight on currently uncovered vertices of X (given by Y) as possible without using an already used A vertex (used A vertices are those outside B).

Theorem 8.3. *Let $\mathcal{H} = (A \cup X, \mathcal{E})$ be a bipartite hypergraph with w a non-negative weight function on X-vertices. Suppose that for some $c > 0$ we have $f_w(Y) \geq cw(Y)$ for all $Y \subset X$. Then the set of edges obtained from the above greedy method is an A-side matching for $\mathcal{H}$ which covers X-vertices with total weight at least $\frac{c}{2+c}w(X)$.*

Proof. Edges are added to $\mathcal{U}$ so as to have A vertices coming from B, and since we delete from B previously added edges to $\mathcal{U}$, this results in $\mathcal{U}$ having unique A vertices for each of its edges. So $\mathcal{U}$ is at every stage in the algorithm an A-side matching. Also note that $\mathcal{U}$ grows throughout, so it suffices to show that at some point in the algorithm $\mathcal{U}$ covers a set of X-vertices with total weight at least $\frac{c}{2+c}w(X)$.

Consider the beginning of the i^{th} loop of the greedy algorithm. Let the set of vertices in X covered so far by $\mathcal{U}$ be equal to $C := X \cap (\cup_{E \in \mathcal{U}} E)$. We have that Y was formed from X by deleting the X vertices from each of the edges in $\mathcal{U}$, so X is the disjoint union of Y with C and $w(X) = w(Y) + w(C)$. Let $w(C) = \alpha w(X)$ for some $\alpha \in [0, 1]$, then $w(Y) = (1 - \alpha)w(X)$. For $j < i$ let $v_j := w(E_j \cap Y_j)$ where $Y_j \supset Y$ was Y at the start of the j^{th} loop. Since $w(E_j \cap Y_j)$ was maximum we have, for any edge E hitting the A vertex of E_j, that $w(E \cap Y) \leq w(E \cap Y_j) \leq v_j$. From construction we have $E_j \cap Y_j$'s disjoint and other X vertices of E_j are already appearing in C at each stage, so $w(C) = \sum_{j<i} v_j$. Note that $A \setminus B$ consists of all A vertices that appear in E_j for some $j < i$, so we have the

following:

$$f_w(Y) = \sum_{a\in A} \max_{E\in\mathcal{E}_a} w(E\cap Y) \tag{8.17}$$

$$\leq \sum_{j<i} v_j + \sum_{a\in B} \max_{E\in\mathcal{E}_a} w(E\cap Y) \tag{8.18}$$

$$= w(C) + \sum_{a\in B} \max_{E\in\mathcal{E}_a} w(E\cap Y) \tag{8.19}$$

$$= \alpha w(X) + \sum_{a\in B} \max_{E\in\mathcal{E}_a} w(E\cap Y). \tag{8.20}$$

From hypothesis we have $f_w(Y) \geq cw(Y) = c(1-\alpha)w(X)$, so combining yields inequalities:

$$\alpha w(X) + \sum_{a\in B} \max_{E\in\mathcal{E}_a} w(E\cap Y) \geq c(1-\alpha)w(X) \tag{8.21}$$

$$\sum_{a\in B} \max_{E\in\mathcal{E}_a} w(E\cap Y) \geq (c-(c+1)\alpha)w(X). \tag{8.22}$$

Therefore there exists an edge E hitting a vertex of B such that:

$$w(E\cap Y) \geq \frac{(c-(c+1)\alpha)}{|B|}w(X) \geq \frac{(c-(c+1)\alpha)}{|A|}w(X). \tag{8.23}$$

So by E_i having maximum $w(E\cap Y)$ over all edges E intersecting B, we will have:

$$w(E_i\cap Y) \geq \frac{(c-(c+1)\alpha)}{|A|}w(X). \tag{8.24}$$

If $w(C) = \alpha w(X) \geq \frac{c}{2+c}w(X)$ the set of edges $\mathcal{U}$ already covers enough weight of X vertices so that the result holds. Otherwise we have that $\alpha < \frac{c}{2+c}$ and therefore:

$$c-(c+1)\alpha > c-(c+1)\frac{c}{2+c} = \frac{c}{2+c}. \tag{8.25}$$

So combining equations 8.24 and 8.25 we have that $v_i = w(E_i\cap Y) > \frac{c}{(2+c)|A|}w(X)$ additional weight of X vertices is added to $\mathcal{U}$ when we conclude the i^{th} loop. If this relationship holds for each of the $m = |A|$ iterations then we have that the final weight on the X-vertices covered is:

$$\sum_{i\in\{1,\ldots,m\}} v_i > |A|(\frac{c}{(2+c)|A|}w(X)) = \frac{c}{2+c}w(X). \tag{8.26}$$

So the result holds in this case. On the other hand we have seen that if $v_i > \frac{c}{(2+c)|A|}w(X)$ fails to hold at any iteration then $w(C) = \alpha w(X) \geq \frac{c}{2+c}w(X)$ which also gives the result. Therefore the greedy algorithm indeed produces an A-side matching which covers X-vertices of total weight at least $\frac{c}{2+c}w(X)$. □

Corollary 8.4. *Let $\mathcal{H} = (A \cup X, \mathcal{E})$ be a bipartite hypergraph satisfying (*) with $c > 0$. Set w to be a weight function on X that is uniformly 1. The set of edges obtained from the greedy algorithm applied to $\mathcal{H}$ and w is an A-side matching for $\mathcal{H}$ which covers at least $\frac{c}{2+c}|X|$ vertices of X.*

Proof. Note that for all $Y \subset X$ we have $w(Y) = |Y|$, so (*) is equivalent to $f_w(Y) \geq cw(Y)$ for all $Y \subset X$. Therefore the greedy algorithm yields an A-side matching covering X-vertices with total weight at least $\frac{c}{2+c}w(X)$, hence covering at least $\frac{c}{2+c}|X|$ vertices of X. □

So in particular when the necessary condition 8.14 holds, that is (*) holds with $c = 1$, we have that there exists an A-side matching which covers at least $\frac{1}{3}$ of the vertices of X. Note that as c approaches ∞ this condition (*) guarantees that a fraction of X vertices arbitrarily close to 100 percent may be covered by an A-side matching.

To show condition 8.16 for any c and any weight function w does not suffice to have an A-side matching covering X we consider for $m, n \in \mathbb{Z}_{\geq 1}$ the following hypergraph $\mathcal{H}_{m,n}$:

Definition 8.5. Let $m, n \in \mathbb{Z}_{\geq 1}$. We define $A := \{1, \ldots, n\}$ and $X := \{1, \ldots, m\}^n$ to be a set of lists with n entries each taking m possible values. For each $a \in A$ and $k \in \{1, \ldots, m\}$ we set $E_{a,k}$ to be the set of lists in X which have a^{th} entry taking value k. Consider the set of hyperedges $\mathcal{E} := \{\{a\} \cup E_{a,k} : a \in A, k \in \{1, \ldots, m\}\}$, then we define the bipartite hypergraph $\mathcal{H}_{m,n} := (A \cup X, \mathcal{E})$.

Consider $\mathcal{H}_{m,n} = (A \cup X, \mathcal{E})$ and notice that $|X| = m^n$. Note that for all $a \in A$ we have m edges of $\mathcal{H}_{m,n}$ hitting a, each of form $\{a\} \cup E_{a,k}$ for some $k \in \{1, \ldots, m\}$. So $\mathcal{E}_a = \{\{a\} \cup E_{a,k} : k \in \{1, \ldots, m\}\}$ for all $a \in A$. Note also that for each $a \in A$ we have that $\{E_{a,k} : k \in \{1, \ldots, m\}\}$ partitions lists in X by their a^{th} entry into m sets of equal size m^{n-1}. So all edges of $\mathcal{H}_{m,n}$ have same size $1 + m^{n-1}$.

Proposition 8.6. *Condition 8.16 holds for $\mathcal{H}_{m,n}$ with constant $c = \frac{n}{m}$ and any non-negative weight function w on X. That is for any non-negative weight function w on X, we have $f_w(Y) \geq \frac{n}{m}w(Y)$ for all $Y \subset X$.*

Proof. Fix w as some non-negative weight function on X. Let $Y \subset X$ and $a \in A$. We have that $\{E_{a,k} \cap Y : k \in \{1, \ldots, m\}\}$ partitions Y into m sets.

So $w(Y) = \sum_{k \in \{1,\ldots,m\}} w(E_{a,k} \cap Y)$ and thus we have some $k_a \in \{1, \ldots, m\}$ such that $w(E_{a,k_a} \cap Y) \geq \frac{w(Y)}{m}$.

So $\max_{E \in \mathcal{E}_a} w(E \cap Y) \geq \frac{w(Y)}{m}$ for all $a \in A$ and therefore:

$$f_w(Y) = \sum_{a \in A = \{1,\ldots,n\}} \max_{E \in \mathcal{E}_a} w(E \cap Y) \geq \frac{n}{m} w(Y). \tag{8.27}$$

□

Proposition 8.7. *Let $\mathcal{U}$ be a set of edges of $\mathcal{H}_{m,n} = (A \cup X, \mathcal{E})$ which contains exactly one edge hitting each vertex of A, then $\mathcal{U}$ covers exactly $m^n - (m-1)^n$ vertices of X.*

Proof. Let $\mathcal{U}$ satisfy the hypothesis with edge hitting each $a \in A$ given by $\{a\} \cup E_{a,k_a}$. The set of vertices in X covered by $\mathcal{U}$ is given by $\bigcup_{a \in A} E_{a,k_a}$. Therefore $x \in X$ is covered by $\mathcal{U}$ if and only if $x \in E_{a,k_a}$ for some $a \in A$. So $x \in X$ is covered by $\mathcal{U}$ if and only if for some $a \in A$ we have that x is a list with a^{th} entry k_a. Note that violating this condition for list $x \in X$ to be covered is equivalent to $x \in \prod_{a \in A}(\{1, \ldots, m\} \setminus \{k_a\})$ being in the Cartesian product over the sets of choices for a^{th} entries disagreeing with k_a, which is a set of size $(m-1)^n$. Therefore $\mathcal{U}$ covers exactly $m^n - (m-1)^n$ vertices of X. □

Corollary 8.8. *Any A-side matching for $\mathcal{H}_{m,n}$ covers at most $(1 - (1 - \frac{1}{m})^n)|X|$ vertices of $|X|$.*

Proof. Since $|X| = m^n$, we have $(1 - (1 - \frac{1}{m})^n)|X| = m^n - (m-1)^n$. So this result follows immediately from the preceding proposition as any A-side matching may be enlarged to have edges hitting each $a \in A$. □

Theorem 8.9. *For any integer $c \geq 1$ and $d < \frac{1}{e^c}$ there exists a bipartite hypergraph $\mathcal{H} = (A \cup X, \mathcal{E})$, which satisfies condition 8.16 with constant c for any non-negative weight function on X, such that no A-side matching can cover more than $(1-d)|X|$ of the vertices in X.*

Proof. Suppose we have c, d as above. From fact that $\lim_{t \to \infty}(1 - \frac{1}{t})^{tc} = \frac{1}{e^c}$ we can pick a positive integer m so that $(1 - \frac{1}{m})^{mc} > d$. Consider bipartite hypergraph $\mathcal{H} := \mathcal{H}_{m,mc} = (A \cup X, \mathcal{E})$. From Proposition 8.6 we have that $\mathcal{H}$ satisfies condition 8.16 with constant $\frac{mc}{m} = c$ for any non-negative weight function w on X.

From Corollary 8.8 any A-side matching for $\mathcal{H}$ covers at most $(1-(1-\frac{1}{m})^{mc})|X|$ vertices in X. We have $(1 - \frac{1}{m})^{mc} > d$, so any A-side matching for $\mathcal{H}$ covers at most $(1-d)|X|$ vertices in $|X|$. Therefore $\mathcal{H}$ is as desired. □

This result shows that for any absolute constant c, having condition 8.16 hold for $\mathcal{H} = (A \cup X, \mathcal{E})$ with every non-negative weight function on X does not suffice to guarantee having an A-side matching which covers more than the portion $(1 - \frac{1}{e^c})|X|$ of the X vertices. Between this and the Corollary 8.4 obtained for the greedy algorithm, we have that condition (*) for value c provides a guarantee for the existence of an A-side matching covering at least a $\frac{c}{2+c}$ fraction of the X vertices, but no better a guarantee than a $1 - \frac{1}{e^c}$ fraction. Tightening these bounds, and related weighted variants, provides an interesting problem to study in bipartite hypergraph covers.

Also of considerable interest are possible strengthenings of condition 8.14 to a sufficient condition for an A-side matching. We proceed to show by a simple application of Hall's Theorem that a strengthening of 8.14 by a linear factor of the maximum edge size does suffice as a sufficient condition for having an A-side matching covering X.

Theorem 8.10. *Let $\mathcal{H} = (A \cup X, \mathcal{E})$ be a bipartite hypergraph with maximum edge size r which satisfies condition (*) with value $c = r - 1$. Then $\mathcal{H}$ admits an A-side matching which covers X.*

Proof. Consider an associated bipartite graph $G = (A \cup X, F)$ with edge set F determined by adjacency rule that for all $a \in A$ and $x \in X$ we have a adjacent to x if and only if there exists hyperedge $E \in \mathcal{E}$ such that $\{a, x\} \subset E$. Let Y be a subset of X. Note that a vertex $a \in A$ is a neighbour to some vertex of Y in G if and only if there exists a hyperedge $E \in \mathcal{E}$ such that $a \in E$ and $E \cap Y \neq \emptyset$. We have from (*) that:

$$f(Y) \geq (r-1)|Y| \tag{8.28}$$

$$\sum_{a \in A} max_{E \in \mathcal{E}_a} |E \cap Y| \geq (r-1)|Y|. \tag{8.29}$$

Since the maximum hyperedge size is r, we have that Y can intersect any edge on at most $r-1$ vertices, so each term in the summation on the left-hand side is at most $r-1$. Therefore there are at least $\frac{r-1}{r-1}|Y| = |Y|$ nonzero terms on the left-hand side of equation 8.29. So there exist at least $|Y|$ vertices of A with a hyperedge of $\mathcal{H}$ hitting a and intersecting Y, hence at least $|Y|$ neighbours to Y in bipartite graph G. Therefore by Hall's Theorem (Theorem 4.7) we have an X-saturating matching M in G. For each edge between $a \in A$ and $x \in X$ in matching M we choose one hyperedge E_a of $\mathcal{H}$ (from the adjacency definition of G) so that $\{a, x\} \subset E_a$. Since M is a matching this selects at most one hyperedge, uniquely identified as E_a, for each $a \in A$. Thus we form a collection of hyperedges $\mathcal{M} := \{E_a : \ a \in A \text{ is saturated by } M\}$ with at most one edge hitting each $a \in A$. Since the matching M saturates X the collection of hyperedges $\mathcal{M}$ covers X as each edge of M is contained in the corresponding hyperedge of $\mathcal{M}$. Therefore $\mathcal{M}$ is an A-side matching covering X as desired. □

Also note that unlike before, as in the condition from Haxell's Theorem, strengthening by a linear factor of maximum edge size restricts us to the setting where X has size at most within a fixed factor of A. This is quite a restrictive setting, especially when one thinks of X as representing a large number of jobs to be assigned among relatively few machines A. So while an absolute constant factor alteration (*) fails to provide a sufficient condition for having an A-side matching which covers X, we are interested in what sort of sublinear function of maximum edge size can serve as a sufficient factor to strengthen the necessary condition 8.14 to a sufficient one. Note that in our examples $\mathcal{H}_{m,n}$ that did not admit A-side matchings covering X, we have edge sizes exponential in terms of the value $\frac{n}{m}$ for which $\mathcal{H}_{m,n}$ satisfies (*), so it does not rule out hope for sublinear factor sufficient conditions.

Chapter 9

Conclusion

In this thesis we have surveyed many of the results concerning polytime approximation and optimal value estimation for the Santa Claus problem and the machine scheduling problem. In all but very special cases, gaps remain between the best known factors for approximation or optimal value estimation and the NP hardness bounds of 2 [6] and $\frac{3}{2}$ [23] for estimation of the Santa Claus and machine scheduling problems respectively. We have seen considerable progress in recent years (see [28, 19] for machine scheduling and [18, 9, 11] for the Santa Claus problem) in narrowing some of these gaps for the restricted cases, but the initial approximation factor of 2 [23] for the machine scheduling problem has been unimproved upon in the restricted case. Improving that approximation factor is a major open problem for scheduling theory.

In some cases, in particular the restricted machine scheduling case [28, 19] we explore at length in Chapter 5, we have seen the use of local search methods utilizing the CLP that also give rise to gaps between best known factors of approximation and optimal value estimation. The refinement of these methods to yield improved factors for approximation remains a major task for research in this area. Even to bring the approximation factor for the restricted machine scheduling problem closer to the best known estimation factor of $\frac{11}{6}$ [19] would constitute a major breakthrough in progress. For the non-restricted case we still lack any constant factor approximation or estimation algorithm for the Santa Claus problem and the CLP is ill-equipped to deal with it, given its unbounded integrality gap in this more general case.

While the approximation factor for the directed (or unrelated) graph balancing case of the Santa Claus problem has been resolved [7, 29], the general approximation factor of 2 has not yet been bested in the directed graph balancing case for machine scheduling. Having a directed graph structure to work with and the rotation algorithm for directed cycles we discuss, may make this seem a more approachable special case, however the CLP integrality gap of 2 [29] indicates that progress here may prove more difficult than for the general restricted case. The graph balancing machine scheduling problem resides in an intermediate state of progress for approximation, with a factor of $\frac{7}{4}$ [12] besting the general 2 approximation factor (and the $\frac{11}{6}$ restricted estimation factor), but not matching the $\frac{3}{2}$ bound. It also has a slightly improved estimation factor [20] which suggests that

algorithmic improvements may be within reach for graph balancing.

Ideas relating to a matroid generalization of the Santa Claus problem were seen here [11] to be useful in application to approximation for the restricted case. In this thesis we have suggested a related machine scheduling variant and propose its properties be investigated, both as a curiosity in its own right and for any potential application to handling the standard cases.

Given the successful application of ideas relating to Haxell's Theorem 4.8 [15] to the Santa Claus problem, we also wonder if thinking generally about one sided matching covers for bipartite hypergraphs can relate to progress on the machine scheduling problem. Along these lines this paper introduces conditions on bipartite hypergraphs (see equations 8.15 and 8.16) that connect to and generalize conditions arising from the machine scheduling CLP. Some areas for further improvements related to these investigations are introduced at the end of Chapter 8, including closing the bounds for the fraction of coverage that is guaranteed by a condition of form (8.15).

There has been a good deal of activity on these two optimization problems over the past 15 years, and indeed much activity in the past year, and there is good reason for optimism with respect to future developments. It is my hope that this thesis has served to introduce these problems and the techniques that have been applied to their study in a manner accessible to a more general audience, and to advance some related ideas that may be of interest to future research.

References

[1] Ron Aharoni, Eli Berger, and Ran Ziv. Independent systems of representatives in weighted graphs. *Combinatorica*, 27(3):253–267, 2017.

[2] Chidambaram Annamalai. Algorithmic advances in allocation and scheduling. *PhD dissertation, ETH Zurich*, 2017.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 2009.

[4] Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms*, 8(3):Article 24, 2012.

[5] Nikhil Bansal and Maxim Sviridenko. The santa claus problem. *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 31–40, 2006.

[6] Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *SIGecom Exchanges*, 5(3), 2005.

[7] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. *Computing Research Repository*, arXiv: 0901.0205, 2009.

[8] Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. On $(1, \epsilon)$-restricted assignment makespan minimization. *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1087–1101, 2015.

[9] Siu-Wing Cheng and Yuchen Mao. Restricted max-min allocation: Approximation and integrality gap. *Computing Research Repository*, arXiv:1905.0608, 2019.

[10] Stephen Cook. The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of Computing*, pages 151–158, 1971.

[11] Sami Davies, Thomas Rothvoss, and Yihao Zhang. A tale of santa claus, hypergraphs and matroids. *Computing Research Repository*, arXiv:1807.07189, 2019.

[12] Tomáš Ebenlendr, Marik Krčál, and Jiří Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *ACM-SIAM Annual Symposium on Discrete Algorithms (SODA)*, pages 483–490, 2008.

[13] Uriel Feige. On estimation algorithms versus approximation algorithms. *Foundations of Software Technology and Theoretical Computer Science (Bangalore)*, 2:357–363, 2008.

[14] Alessandra Graf and Penny Haxell. Finding independent transversals effectively. *Computing Research Repository*, arXiv: 1811.02687, 2018.

[15] Penny Haxell. A condition for matchability in hypergraphs. *Graphs and Combinatorics*, 11:245–248, 1995.

[16] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the Association for Computing Machinery*, 34(1):144–162, 1987.

[17] Chien-Chung Huang and Sebastian Ott. A combinatorial approximation algorithm for graph balancing with light hyper edges. *Proceedings of the European Symposium on Algorithms (ESA)*, 2016.

[18] Klaus Jansen and Lars Rohwedder. A note on the integrality gap of the configuration lp for restricted santa claus. *Computing Research Repository*, arXiv: 1807.03626, 2016.

[19] Klaus Jansen and Lars Rohwedder. On the configuration-lp of the restricted assignment problem. *Computing Research Repository*, arXiv: 1611.01934, 2016.

[20] Klaus Jansen and Lars Rohwedder. Local search breaks 1.75 for graph balancing. *Computing Research Repository*, arXiv:1811.00955, 2018.

[21] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[22] Leonid Khachiyan. A polynomial-time algorithm for solving linear programs. *Mathematics of Operations Research*, 5(1), 1980.

[23] Jan Karol Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

[24] Leonid A. Levin. Universal sequential search problems. *Peredachi Informatsii*, 9(3):115–116, 1973.

[25] James G. Oxley. *Matroid Theory*. Oxford University Press, New York, NY, USA, 1992.

[26] Daniel R. Page and Roberto Solis-Oba. A 3/2-approximation algorithm for the graph balancing problem with two weights. *Algorithms*, 9(2):38, 2016.

[27] Petra Schuurman and Gerhard J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, pages 203–213, 1999.

[28] Ola Svensson. Santa claus schedules jobs on unrelated machines. *SIAM Journal on Computing*, 41(5):1318–1341, 2012.

[29] José Verschae and Andreas Wiese. On the configuration lp for scheduling on unrelated machines. *Journal of Scheduling*, 17(4):371–383, 2014.

[30] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 2011.