ALGORITHMS FOR INTEGER PROGRAMMING AND ALLOCATION

LARS ROHWEDDER

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Technische Fakultät der Christian-Albrechts-Universität zu Kiel

September 2019

1. GUTACHTER Prof. Dr. Klaus Jansen

2. GUTACHTER Prof. Dr. Ola Svensson

3. GUTACHTER Prof. Dr. Roberto Solis-Oba

DATUM DER DISPUTATION 25. September 2019

There once lived a man who learned how to slay dragons
and gave all he possessed to mastering the art.
After three years he was fully prepared but, alas,
he found no opportunity to practise his skills.

— Zhuang Zhou, philosopher

As a result he began to teach how to slay dragons.

— René Thom, Fields medalist

For Faiz.

ABSTRACT

The first part of the thesis contains pseudo-polynomial algorithms for integer linear programs (ILP). When certain parameters of an ILP are fixed, that is, they are treated as constants in the running time, it is possible to obtain algorithms with a running time that is pseudo-polynomial in the entries of the ILP's matrix. We present a tight pseudo-polynomial running time for ILPs with a constant number of constraints. Furthermore, we study an extension of this model to MILPs (linear programs that contain both fractional and integer variables). Then we move to n-fold ILPs, a class of ILPs with block structured matrices. We present the first algorithm for n-folds, which is near-linear in the dimensions of the ILP.

The second part is about scheduling in non-identical machine models, more precisely, restricted allocation problems. Here a set of jobs has to be allocated to a set of machines. However, every job has a subset of machines and may only be assigned to a machine from this subset. We consider the objectives of minimizing the makespan or maximizing the minimum load. We study the integrality gap of a particularly strong linear programming relaxation, the configuration LP, for variations of this problem. The integrality gap can be seen as a measure of strength of an LP relaxation. A local search technique can be used to bound this value. However, the proofs are generally non-constructive, i.e., they do not give an efficient approximation algorithm right away. We derive better upper bounds on the integrality gap of the problems Restricted Assignment, Restricted Santa Claus, and Graph Balancing. Furthermore, we give the first (constructive) quasi-polynomial time approximation algorithm for Restricted Assignment with an approximation ratio strictly less than 2.

ZUSAMMENFASSUNG

Der erste Teil der Thesis umfasst pseudopolynomielle Algorithmen für ganzzahlige lineare Programme (ILP). Wenn bestimmte Parameter eines ILPs fixiert sind, d.h. sie werden in der Laufzeit als Konstanten betrachtet, dann ist es möglich Algorithmen zu entwerfen, deren Laufzeit pseudopolynomiell in dem größten absoluten Wert eines Eintrags der Matrix des ILPs ist. Ein Ergebnis, das wir präsentieren, ist eine scharfe Schranke für die pseudopolynomielle Laufzeit, die nötig ist um ein ILP mit konstant vielen Bedingungen zu lösen. Danach befassen wir uns mit n-fold ILPs, eine Klasse von ILPs, deren matrix eine Blockstruktur besitzt. Wir geben den ersten Algorithmus für n-folds an, dessen Laufzeit gleichzeitig nahezu linear in der Dimension des ILPs ist.

Der zweite Teil handelt von nicht-identischen (heterogenen) Maschinen Modellen, genauer gesagt *restricted allocation problems*. Hier soll eine Menge von Jobs auf eine Menge von Maschinen verteilt werden. Jeder Job darf aber nur auf bestimmte Maschinen zugewiesen werden. Wir betrachten als Zielfunktionen sowohl die Minimierung des Makespans als auch die Maximierung der minimalen Last einer Maschine. Wir untersuchen den *integrality gap* einer besonders starken LP Relaxierung, dem Konfigurations LP, für Variationen dieses Problems. Der *integrality gap* kann als Maß für die Stärke einer LP Relaxierung gesehen werden. Über ein Argument mittels einer lokalen Suche wird dieser Wert beschränkt. Jedoch sind die Beweise typischerweise nicht konstruktiv, d.h. sie implizieren nicht direkt effiziente Approximationsalgorithmen. Wir beweisen neue obere Schranken an den *integrality gap* für die Probleme Restricted Assignment, Restricted Santa Claus und Graph Balancing. Desweiteren präsentieren wir den ersten (konstruktiven) Quasipolynomialzeit Approximationsalgorithmus für das Restricted Assignment Problem mit Approximationsrate echt kleiner als 2.

PUBLICATIONS

Most of the results presented in this thesis are based on the following peer-reviewed publications. They are fully contained in this this thesis, except for [44] which is used only in parts.

[39] Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. "Near-Linear Time Algorithm for n-fold ILPs via Color Coding." In: *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*. 2019, 75:1–75:13.

[40] Klaus Jansen and Lars Rohwedder. "A Quasi-Polynomial Approximation for the Restricted Assignment Problem." In: *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization, IPCO'17, Waterloo, ON, Canada, June 26-28, 2017*. 2017, pp. 305–316.

[41] Klaus Jansen and Lars Rohwedder. "On the Configuration-LP of the Restricted Assignment Problem." In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'17, Barcelona, Spain, January 16-19*. 2017, pp. 2670–2678.

[44] Klaus Jansen and Lars Rohwedder. "Compact LP Relaxations for Allocation Problems." In: *1st Symposium on Simplicity in Algorithms, SOSA'18, January 7-10, 2018, New Orleans, LA, USA*. 2018, 11:1–11:19.

[45] Klaus Jansen and Lars Rohwedder. "Local Search Breaks 1.75 for Graph Balancing." In: *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*. 2019, 74:1–74:14.

[46] Klaus Jansen and Lars Rohwedder. "On Integer Programming and Convolution." In: *10th Innovations in Theoretical Computer Science Conference, ITCS'19, January 10-12, 2019, San Diego, California, USA*. 2019, 43:1–43:17.

Moreover, the following report is made publicly available and is intended to appear in a peer-reviewed conference or journal.

[43] Klaus Jansen and Lars Rohwedder. "A note on the integrality gap of the configuration LP for restricted Santa Claus." In: *CoRR* abs/1807.03626 (2018).

ACKNOWLEDGMENTS

I wish to thank Faiz for her support and encouragement. I am grateful for my family and friends, in particular, my parents Thomas and Gisela.

I would like to offer my special thanks to my adviser Klaus Jansen. Thanks also to my other co-authors in the past: Sebastian Berndt, Marin Bougeret, Leah Epstein, Alexandra Lassota, Asaf Levin, and Marten Maack. My regards to the fantastic group in Kiel, with whom I had countless exciting discussions, collaborations, and trips: Sebastian Berndt, Max Deppert, Kilian Grage, Ute Iaquinto, Klaus Jansen, Kim Klein, Felix and Kati Land, Alexandra Lassota, Marten Maack, Parvaneh Massouleh, Marcin Pal, and Malin Rau.

Finally, I wish to thank those who hosted me for research visits, took the time to introduce me to their country and to exchange ideas: Marin Bougeret, Leah Epstein, Asaf Levin, Michael Poss, Ola Svensson, and José Verschae. I had the pleasure to travel to many overwhelming places for my Ph.D. work and I feel very fortunate for this opportunity.

CONTENTS

1 INTRODUCTION

The problems studied in theoretical computer science are often inspired by practical applications. For instance, a frequent question for a researcher is how to spend his travel budget. He can go to different conferences, workshops, or go to other universities in order to exchange ideas. Unfortunately, his funding is limited and he cannot do all trips. A theoretician would forget about the specifics of the trips—whether they are conferences or workshops; whether he goes by train or aircraft. He would model them as simple items with a cost, which is the price of the trip, and a value, which is a numerical assessment of its importance. We assume for simplicity that money is the only constraint and no two trips are in conflict with each other. This leaves the researcher with a very clean mathematical definition of the problem.

How can we solve such a problem? In *Combinatorial Optimization* for a fixed input the number of candidates for a solution is usually finite and can be enumerated. For our travel budget problem, every solution has to be a subset of items. This makes 2^n candidates for n items. A simple algorithm could enumerate them all, for each check if it fits the budget, and take the most valuable among them. This is not an efficient algorithm. If checking a candidate takes one millisecond, then for $n = 50$ items this would take about 35 thousand years. But how can we formally distinguish efficient from inefficient? Clearly, for an input with more items an algorithm will require more time to find a solution than for fewer items. Hence, we measure the running time of an algorithm with respect to the encoding length of the input, i.e., how many bits it takes to encode the input. Although a controversial topic, in theory the usual notion of efficient is a polynomial running time. This means the algorithm terminates in time $|I|^C$ where $|I|$ is the encoding size of the input and C is some constant. On the other hand, an exponential running time, e.g., $C^{|I|}$, is usually considered inefficient. This renders the enumeration algorithm above as inefficient.

Many problems in Combinatorial Optimization behave similar to the one above: A solution has polynomial size and can be checked efficiently. This a class is known as NP [1]. It is not known whether all of them are in P, the class of problems which can be solved efficiently, that is, in polynomial time. While for some problems efficient algorithms were found, others (like the travel budget problem) have turned out to be notoriously hard. This has led to the widely believed P ≠ NP conjecture, which states that not all problems in NP are solvable in polynomial time. The conjecture is one of the Millennium Prize Problems, a list of arguably the most important open questions in math.

1 Technically this refers to the decision variant of the problems: *Is there a solution of value at least (or at most) T for a given T?*

Regardless how far science might be from answering this question, surprisingly for some problems in NP we know that they cannot be solved in polynomial time, if P ≠ NP holds. This is the class of NP-complete problems. Although NP-complete problems might appear hopeless at first sight, they do offer intriguing research directions.

APPROXIMATION ALGORITHMS. It is often possible to produce a good solution—though not always optimal—in polynomial time. Perhaps some selection of trips is not the best and the value is 10% below the optimal selection, but this might still be acceptable. Let $\mathrm{OPT}(I)$ denote the optimal value of instance I of some maximization problem, that is, we are looking for a solution of maximal value. Let $A(I)$ be the value of the solution an algorithm A produces for instance I. We say, A is a *c-approximation algorithm*, if for all instances I, $A(I) \geq 1/c \cdot \mathrm{OPT}(I)$. Similarly, for a minimization problem, we call it a c-approximation algorithm, if $A(I) \leq c \cdot \mathrm{OPT}(I)$ for all I. Typically, we are interested in c-approximation algorithms, where c is a constant. When a problem admits such an algorithm, we usually want to determine the smallest c for which there exists a polynomial time c-approximation.

PARAMETERIZED AND PSEUDO-POLYNOMIAL ALGORITHMS. When a certain parameter k in the input is small, it might be easier to solve a problem optimally. In the travel budget problem one could for example consider the case where only k different costs appear, that is, many items have the same costs. One may hope to find optimal algorithms for NP-complete problems where the running time depends exponentially on some parameter, but not on the total input size. For example, such a running time could look like $|I|^k$ or even $2^k \cdot |I|$. An algorithm with a running time of the latter type, where the exponent of $|I|$ is a constant independent of k, but an arbitrary function in k is allowed as a multiplicative factor, is particularly favorable and is called *fixed-parameter tractable (FPT)*.

Another related term is pseudo-polynomial running time. Suppose in the input is some natural number N—like the budget in our example. Then encoding this number takes $O(\log(N))$ bits. Therefore, a running time of the form $O(N)$ could already be exponential in the input size. Like for a parameter k, we sometimes allow algorithms that have a higher dependency on some number. If it is polynomial in N, but not necessarily in $\log(N)$, we say the running times is *pseudo-polynomial* in N.

A particularly fascinating problem in Optimization is that of *(integer) linear programming*, because it easily models a large number of problems. A linear program consists of n positive real variables $x = (x_1, \ldots, x_n)^T$ and a linear objective function $c \in \mathbb{R}^n$. We are looking for a solution that maximizes $c_1 x_1 + \cdots + c_n x_n$. We also allow m linear constraints of the form $a_{j,1} x_1 + \cdots + a_{j,n} x_n = b_j$ on the variables. They can also be written in matrix terms as $Ax = b$, where $A = (a_{i,j}) \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Although there are some

other, mostly equivalent, variations, using for example inequalities instead of equalities, we mostly study the standard form

$$\begin{aligned} \max\ & c^T x \\ & Ax = b \\ & x_i \geq 0 \qquad\qquad \forall i \in \{1,\dots,n\} \end{aligned}$$

Linear programs (LPs) can be solved efficiently both in theory using the ellipsoid method and in practice using the simplex algorithm (see for example [63]). Many real world problems can be modeled directly as a linear program and therefore they can also be solved efficiently. However, discrete problems often need another feature, which is to be able to model integer variables.

For example, the travel budget problem could be modeled with two variables $x_i, \overline{x}_i$ for each item i and one variable r that describes the remaining budget. We write c_i for the cost of item i and p_i for its profit. By b we denote the budget. Consider the system

$$\begin{aligned} \max\ & p_1x_1 + \cdots + p_nx_n \\ & c_ix_1 + \cdots c_nx_n + r = b \\ & x_i + \overline{x}_i = 1 \qquad\qquad \forall i \\ & x_i, \overline{x}_i \geq 0 \text{ integer} \qquad\qquad \forall i \\ & r \geq 0 \text{ integer}\,. \end{aligned}$$

This models the problem: For each item i when $x_i = 1$, we select the item; when $\overline{x}_i = 1$, we do not. The constraint $x_i + \overline{x}_i = 1$ enforces that exactly one of x_i and $\overline{x}_i$ is 1; the objective $p_1x_1 + \cdots + p_nx_n$ is exactly the profit of the solution; and the constraint $c_ix_1 + \cdots c_nx_n + r = b$ guarantees that the budget suffices. It is often impossible to cope with discrete problems like this without using integer variables. Unfortunately, integer linear programs (ILPs) are usually hard to solve. Its general form is NP-complete. Hence, we will study parameterized algorithms for them.

Our first contribution is a pseudo-polynomial algorithm for ILPs with a constant number of constraints. The running time improves on state-of-the-art literature and interestingly we also show matching conditional lower bounds on the running time.

Theorem 7. *Let* $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ *be an ILP where* $A \in \mathbb{Z}^{m\times n}$. *We can solve this ILP in time* $O(nm) + O(\sqrt{m}\Delta)^{2m} \cdot \log(\|b\|_\infty)$, *where* Δ *is the biggest absolute value of an entry in* A.

We also study the setting where only feasibility is checked, i.e., there is no objective, and show connections to the problem $(\min,+)$-convolution and to discrepancy theory. We will present these results in Chapter 2.

In Chapter 3 we extend the previous setting to a *mixed integer linear program (MILP)*. These are linear programs that contain both integer and fractional variables. We consider linear programs with m integer variables

$x = (x_1, \ldots, x_m)$ and n fractional variables $y = (y_1, \ldots, y_n)$. There are only r constraints for the integer variables and the coefficients are small, but the number of constraints and coefficients for the fractional variables can be large. Let $A \in \mathbb{Z}^{r \times m}$, $B \in \mathbb{Z}^{r \times n}$, and $C \in \mathbb{Z}^{s \times n}$; let $c \in \mathbb{R}^m$, $d \in \mathbb{R}^n$, and $b \in \mathbb{Z}^{r+s}$. Furthermore, let Δ be an upper bound on the entries in A. We study the linear program

$$\begin{aligned} &\max\ c^T x + d^T y \\ &\begin{pmatrix} A & B \\ 0 & C \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = b \\ &\qquad x_i \geq 0 \text{ integer} && \forall i \in \{1, \ldots, m\} \\ &\qquad y \geq 0 && \forall i \in \{1, \ldots, n\}. \end{aligned}$$

We show that for a fixed r this linear program is again solvable in pseudo-polynomial time in Δ. To our best knowledge, we are the first to study MILPs from this perspective.

Theorem 15. *There is an algorithm that solves the MILP above in time*

$$(r\Delta)^{O(r^2)} \cdot |I|^{O(1)},$$

where $|I|$ is its encoding length.

The running time of the algorithms above explodes when the number of constraints on the integer variables increases. Next we will consider a setting with many constraints on the integer variables, which we can still solve efficiently. In this setting the non-zero entries of the matrix have a nice block structure. Consider a matrix of the form

$$\mathcal{A} = \begin{pmatrix} A_1 & A_2 & \ldots & A_n \\ B_1 & 0 & \ldots & 0 \\ 0 & B_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & B_n \end{pmatrix},$$

where $A_1, \ldots, A_n \in \mathbb{Z}^{r \times t}$ and $B_1, \ldots, B_n \in \mathbb{Z}^{s \times t}$ are each small matrices. We call $\mathcal{A}$ an n-fold matrix. The n-fold integer programming problem is to find the optimum for $\max\{c^T x : \mathcal{A}x = b, \ell \leq x \leq u\}$, where $c, \ell, u \in \mathbb{Z}^{nt}$ are objective function and lower and upper bounds on the variables. If r and s are constants, we can again solve the ILP in pseudo-polynomial time.

Theorem 23. *The n-fold integer programming problem can be solved in time $(rs\Delta)^{O(r^2 s + s^2)} \cdot L^2 \cdot nt \log^{O(1)}(nt)$, where Δ is the largest absolute value in $\mathcal{A}$ and L is the binary encoding length of the largest integer in c, ℓ, u, b.*

There has been a long line of research on this problem, but we present the first algorithm with a near-linear dependency on the number of variables nt.

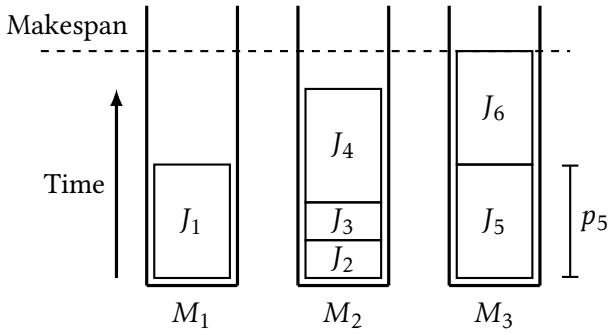


Figure 1: Makespan minimization

This concludes our work on integer programming. We proceed with scheduling problems and move from parameterized algorithm to approximation algorithms.

Let $\mathcal{J} = \{J_1, \dots, J_n\}$ be a set of jobs. Each job J_i has a processing time $p_j > 0$. Moreover, let $\mathcal{M} = \{M_1, \dots, M_m\}$ be a set of machines. At each time a machine can only process one job and when a job is started, it has to be processed completely. A popular objective we consider is the makespan. This is the time when all jobs are completed. Clearly, it does not make sense to have idle times between any two jobs. Hence, it suffices to find an allocation $\sigma : \mathcal{J} \to \mathcal{M}$ and to minimize the maximum load, i.e.,

$$\max_{M_i \in \mathcal{M}} \sum_{\substack{J_j \in \mathcal{J} \\ \sigma(J_j) = M_i}} p_j.$$

See also Figure 1 for illustration. The problem above is called Scheduling on Identical Machines. After preprocessing which includes the rounding of the processing times (inducing a small error), this problem can be modeled as an ILP with small entries and few constraints. Using our result from Chapter 2, one can find a $(1+\epsilon)$-approximation for every $\epsilon > 0$ in polynomial time (PTAS). We give a detailed description of this procedure in said chapter. As the name suggests, all machines in this problem behave identically. In the remainder we study problems with non-identical machines. This usually makes the problems much harder and indeed for all of the problems considered from here on there does not exist a PTAS unless P = NP. The method we use is again a linear program, but this time one which we cannot solve with integral variables. Instead, we study its *continuous relaxation*.

CONTINUOUS RELAXATIONS AND INTEGRALITY GAPS. For sake of brevity, we focus on minimization objectives like the makespan. However, the same approach works also for maximization. The continuous relaxation is the linear program we get by omitting the integrality constraints of an ILP. We let $\mathrm{OPT}(I)$ and $\mathrm{OPT}^*(I)$ denote the integral and fractional (continuous) optimum of the linear program for an instance I. A powerful and widely used approach is to construct an algorithm A that for some constant $c > 1$ always finds an integral solution with value $A(I) \leq c \cdot \mathrm{OPT}^*(I)$. This could for

example be by solving the relaxation and (possibly in a non-trivial way) round the variables to integers afterwards. Using A, we easily get a c-approximation algorithm. Since any integral solution is also a solution for the relaxation, we have $\mathrm{OPT}^*(I) \leq \mathrm{OPT}(I)$. Thus,

$$A(I) \leq c \cdot \mathrm{OPT}^*(I) \leq c \cdot \mathrm{OPT}(I).$$

The prospects of this approach heavily depend on the linear program used. If for example there are instances I with $\mathrm{OPT}(I)/\mathrm{OPT}^*(I) > c$, then there cannot be such an algorithm A (note that $A(I) \geq \mathrm{OPT}(I)$). We define the *integrality gap* of a linear programming relaxation as

$$\sup_I \frac{\mathrm{OPT}(I)}{\mathrm{OPT}^*(I)},$$

i.e., the worst case ratio between integral and continuous optimum. Generally, we are interested in linear programming relaxations that have a small integrality gap. These are considered the strongest. In many cases there is an approximation algorithm like A which implies a bound on the integrality gap. Sometimes, however, we only have a non-constructive proof of an integrality gap. This can be seen as a hint that an approximation algorithm is likely to exist and that this particular linear program is useful. Another reason why the integrality gap is relevant is that it yields an efficient estimation algorithm. When a continuous relaxation has an integrality gap of α and the fractional optimum is $\mathrm{OPT}^*(I)$, then we know that the integral optimum $\mathrm{OPT}(I)$ can be bounded by $\mathrm{OPT}^*(I) \leq \mathrm{OPT}(I) \leq \alpha\mathrm{OPT}^*(I)$. In other words, if we can solve the relaxation, which is usually the case, then we have a good estimation of $\mathrm{OPT}(I)$.

We will now introduce the Restricted Assignment problem and related problems. It is a generalization of Scheduling on Identical Machines. The difference is that not all jobs can be scheduled on every machine. A job J_j has a set $\Gamma(J_j) \subseteq \mathcal{M}$ of machines on which it can be processed. All other machines are forbidden for this particular job. Again, we are minimizing the makespan. We consider a linear programming relaxation called configuration LP. A configuration for a particular machine M_i and a makespan τ is a set of jobs that may be processed in M_i and have a total volume less than τ. We define

$$\mathcal{C}(M_i, \tau) = \left\{ S \subseteq \mathcal{J} \ : \ M_i \in \Gamma(J_j) \ \forall J_j \in S \text{ and } \sum_{J_j \in S} p_j \leq \tau \right\}.$$

An ILP for Restricted Assignment can be constructed by assigning configurations to machines. Each machine has one configuration and each job must appear in one configuration:

$$\sum_{C \in C(M_i,\tau)} x_{i,C} = 1 \qquad \forall M_i \in \mathcal{M}$$

$$\sum_{M_i \in \mathcal{M}} \sum_{\substack{C \in C(M_i,\tau) \\ J_j \in C}} x_{i,C} = 1 \qquad \forall J_j \in \mathcal{J}$$

$$x_{i,C} \geq 0 \text{ integer} \qquad \forall i \in \mathcal{M}, C \in C(M_i,\tau)$$

Here τ has to be minimized. It must be a constant, since the matrix of the ILP depends on its value. Instead of using an objective in the ILP, we define the optimum to be the smallest τ for which it is feasible. There can be an exponential number of configurations. Therefore also the size of the ILP can be exponential. Nevertheless, a $(1+\epsilon)$-approximate solution of its continuous relaxation can be found in polynomial time for every $\epsilon > 0$. There are simpler ILP formulations for this problem, but the configuration LP is the strongest known.

We also study the closely related restricted Santa Claus problem, which is like Restricted Assignment, but we are maximizing the minimum load instead of minimizing the maximum. For this problem the configuration LP can be constructed analogously. The restricted Santa Claus problem is the origin of a local search technique, which lets us bound the integrality gap of the configuration LP. This technique, however, does not easily lead to a polynomial time algorithm. Hence, the bounds are usually non-constructive. We start by giving an improvement of the technique for the restricted Santa Claus problem in Chapter 5.

Theorem 28. *The configuration LP for restricted Santa Claus has an integrality gap of at most* $3 + 5/6 \approx 3.8333$ [2].

The previous best bound was 4. In the first part of Chapter 6 we give an upper bound on the integrality gap of the configuration LP for Restricted Assignment.

Theorem 42. *The configuration LP for Restricted Assignment has an integrality gap of at most* $11/6 \approx 1.8333$.

This improves on a bound of $33/17 \approx 1.9412$. Like in the proof for restricted Santa Claus we use a local search algorithm. This algorithm is guaranteed to find an integral solution with makespan no more than $11/6 \cdot \mathrm{OPT}^*$. In particular, this implies the existence of such a solution. Unfortunately the bound on the running time is exponential. Therefore, we do not get an efficient approximation algorithm. In the remainder of the chapter, we explain how to improve the running time to a quasi-polynomial one.

2 The same bound was found simultaneously and independently by Cheng and Mao [22]. Later the same authors improved it to 3.808 [23].

Theorem 50. *We can find a* $(11/6+\epsilon)$*-approximate solution for* Restricted Assignment *in time* $n^{O(1/\epsilon \log(n))}$ *for every* $\epsilon > 0$*, where* $n = |\mathcal{J}| + |\mathcal{M}|$.

This is the first algorithm for the problem with an approximation rate better than 2 and sub-exponential running time. It remains open, whether this is also possible in polynomial time. A special case of Restricted Assignment is called the Graph Balancing problem. This is the case where $|\Gamma(j)| \leq 2$ for all $j \in \mathcal{J}$. Equivalently, we can consider a weighted undirected (multi-)graph and ask for an orientation of the edges such that the maximum weighted in-degree over all vertices is minimized. The state-of the-art for this problem is a 1.75-approximation algorithm and this value is also the integrality gap of the algorithm's underlying LP relaxation. This relaxation is a simpler formulation than the configuration LP and it is well known that it cannot be stronger. In Chapter 7 we show that the configuration LP is in fact strictly stronger than this LP.

Theorem 51. *The configuration LP for* Graph Balancing *has an integrality gap of at most* 1.749.

Although this is only a marginal improvement of 0.001, it breaks an important barrier.

This chapter is about solving Integer Linear Programs in pseudo-polynomial time when the number of constraints is a constant. A new algorithm and matching conditional lower bounds are presented.

2 PSEUDO-POLYNOMIAL INTEGER PROGRAMMING

Vectors $v^{(1)},\dots,v^{(n)} \in \mathbb{R}^m$ that sum up to 0 can be seen as a circle in $\mathbb{R}^m$ that walks from 0 to $v^{(1)}$ to $v^{(1)}+v^{(2)}$, etc. until it reaches $v^{(1)}+\dots+v^{(n)}=0$ again. The Steinitz Lemma [67] says that if each of the vectors is small with respect to some norm, we can reorder them in a way that each point in the circle is not far away from 0 w.r.t. the same norm.

Recently Eisenbrand and Weismantel found a beautiful application of this lemma in the area of integer programming [30]. They looked at ILPs of the form $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$, where $A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^n$ and obtained a pseudo-polynomial algorithm in Δ, the biggest absolute value of an entry in A, when m is treated as a constant. The running time they achieve is $n \cdot O(m\Delta)^{2m} \cdot \|b\|_1^2$ for finding the optimal solution and $n \cdot O(m\Delta)^m \cdot \|b\|_1$ for finding only a feasible solution. This improves on a classic algorithm by Papadimitriou [61], which has a running time of

$$O(n^{2m+2} \cdot (m \cdot \max\{\Delta, \|b\|_\infty\})^{(m+1)(2m+1)}).$$

The basic idea in [30] is that a solution x^* for the ILP above can be viewed as a walk in $\mathbb{Z}^m$ starting at 0 and ending at b. Every step is a column of the matrix A: For every $i \in \{1,\dots,n\}$ we step x_i^* times in the direction of A_i (see upper picture in Figure 2). By applying the Steinitz Lemma they show that there is an ordering of these steps such that the walk never strays off far from the direct line between 0 and b (see lower picture in Figure 2). They construct a directed graph with one vertex for every integer point near the line between 0 and b and create an edge from u to v, if $v-u$ is a column in A. The weight of the edge is the same as the c-value of the column. An optimal solution to the ILP can now be obtained by finding a longest path from 0 to b. This can be done in the mentioned time, if one is careful with circles.

Our approach does not reduce to a longest path problem, but rather solves the ILP in a divide and conquer fashion. We use the (weaker) assumption that a walk from 0 to b visits a vector b' near $b/2$ at some point. We guess this vector and solve the problem with $Ax = b'$ and $Ax = b - b'$ independently. Both results can be merged to a solution for $Ax = b$. In the sub-problems the norm of b and the norm of the solution are roughly divided in half. We use this idea in a dynamic program and speed up the process of merging solutions using algorithms for convolution problems. This approach gives us

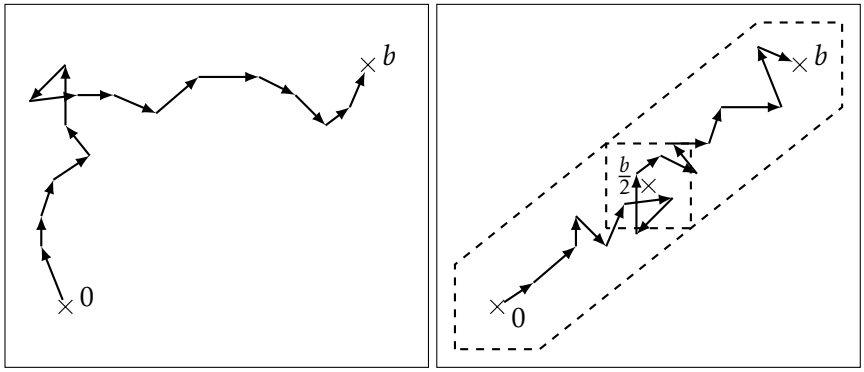


Figure 2: Steinitz Lemma in Integer Programming

better running times for both the problem of finding optimal solutions and for testing feasibility only. We complete our study by giving (almost) tight conditional lower bounds on the running time in which such ILPs can be solved.

Detailed description of results

In the running times we give, we frequently use logarithmic factors like $\log(k)$ for some parameter k. To handle the values $k \in \{0, 1\}$ formally correct, we would need to write $\log(k+1)+1$ instead of $\log(k)$ everywhere. This is ignored for simplicity of notation.

OPTIMAL SOLUTIONS FOR ILPS. We show that a solution to

$$\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$$

can be found in time

$$O(H)^{2m} \cdot \log(\|b\|_\infty) + O(nm) \leq O(\sqrt{m}\Delta)^{2m} \cdot \log(\|b\|_\infty) + O(nm)$$

for a given upper bound H on the hereditary discrepancy[1] of A. For the most part, we will think of H as the general bound of $6\sqrt{m}\Delta$ as given by the Six Standard Deviations Theorem. If we have a vertex solution to the fractional relaxation, we can even get to $O(H)^{2m} + O(nm)$. The running time can be improved if there exists a truly sub-quadratic algorithm for (min, +)-convolution (see Section 2.3 for details on the problem). However, it has been conjectured that no such algorithm exists and this conjecture is the base of several lower bounds in fine-grained complexity [25, 55, 11]. We show that for every m the running time above is essentially the best possible unless the (min, +)-convolution conjecture is false. More formally, for every m there exists no algorithm that solves ILP in time $f(m) \cdot (n^{2-\delta} + (\Delta + \|b\|_\infty)^{2m-\delta})$ for some $\delta > 0$ and an arbitrary computable function f, unless there exists a truly sub-quadratic algorithm for (min, +)-convolution. Indeed, this means there is an equivalence between improving algorithms for (min, +)-convolution and

1 see Preliminaries for definition

for ILPs of fixed number of constraints. It is notable that this also rules out improvements when both Δ and $\|b\|_\infty$ are small. Our lower bound does leave open some trade-off between n and $O(H)^m$ like for example $n \cdot O(H)^m \cdot \log(\|b\|_\infty)$, which would be an interesting improvement for sparse instances, i.e., when $n \ll (2\Delta+1)^m$. A running time of $n^{f(m)} \cdot (\Delta+\|b\|_\infty)^{m-\delta}$, however, is not possible (see feasibility below).

FEASIBILITY OF ILPS. Testing only the feasibility of an ILP is easier than finding an optimal solution. It can be done in time

$$O(H)^m \cdot \log(\Delta) \cdot \log(\Delta + \|b\|_\infty) + O(nm) \\ \leq O(\sqrt{m}\Delta)^m \cdot \log(\Delta) \cdot \log(\Delta + \|b\|_\infty) + O(nm)$$

by solving a Boolean convolution problem that has a more efficient algorithm than the (min, +)-convolution problem that arises in the optimization version. Under the STRONG EXPONENTIAL TIME HYPOTHESIS (SETH) this running time is tight except for logarithmic factors. If this conjecture holds, there is no $n^{f(m)} \cdot (\Delta + \|b\|_\infty)^{m-\delta}$ time algorithm for any $\delta > 0$ and any computable function f.

Other related work

The case where the number of variables n is fixed and not m as here behaves somewhat differently. There is a $2^{O(n\log(n))} \cdot |I|^{O(1)}$ time algorithm ($|I|$ being the encoding length of the input), whereas an algorithm of the kind $f(m) \cdot |I|^{O(1)}$ (or even $|I|^{f(m)}$) is impossible for any computable function f, unless P = NP. This can be seen with a trivial reduction from UNBOUNDED KNAPSACK (where $m = 1$). The $2^{O(n\log(n))} \cdot |I|^{O(1)}$ time algorithm is due to Kannan [48] improving over a $2^{O(n^2)} \cdot |I|^{O(1)}$ time algorithm by Lenstra [47]. It is a long open question whether $2^{O(n)} \cdot |I|^{O(1)}$ is possible instead [30].

Another intriguing question is whether a similar running time as in this work, e.g., $(\sqrt{m}\Delta + \sqrt{m}\|b\|_\infty)^{O(m)} \cdot n^{O(1)}$, is possible when upper bounds on variables are added to the ILP. In [30] an algorithm for this extension is given, but the exponent of Δ is $O(m^2)$.

As for other lower bounds on pseudo-polynomial algorithms for integer programming, Fomin et al. [31] prove that the running time cannot be $n^{o(m/\log(m))} \cdot \|b\|_\infty^{o(m)}$ unless the ETH (a weaker conjecture than the SETH) fails. Their reduction implies that there is no algorithm with running time $n^{o(m/\log(m))} \cdot (\Delta + \|b\|_\infty)^{o(m)}$, since in their construction the matrix A is non-negative and therefore columns with entries larger than $\|b\|_\infty$ can be discarded; thus leading to $\Delta \leq \|b\|_\infty$. Very recently, Knop et al. [53] show that under the ETH there is also no $2^{o(m\log(m))} \cdot (\Delta + \|b\|_\infty)^{o(m)}$ time algorithm. An interesting aspect of this function is that it matches the dependency in m achieved here and in [30] up to a constant in the exponent. Our lower bound differs substantially from the two above. We concentrate on the dependency on Δ and give a precise value of the constant in its exponent.

2.1 PRELIMINARIES

We are assuming a word size of

$$O(m\log(m\Delta)+\log(\|b\|_\infty)+\log(\|c\|_\infty))$$

in the word RAM model, that is to say, arithmetic operations on numbers of this encoding size take constant time. When considering m to be a constant, this makes perfect sense. Also, since we are going to use algorithms with space roughly $O(\sqrt{m}\Delta)^m$, it is only natural to assume that a single pointer fits into a word.

In the remainder of the chapter we will assume that A has no duplicate columns. Note that we can completely ignore a column i, if there is another identical column i' with $c_{i'} \geq c_i$. This implies that in time $O(nm)+O(\Delta)^m$ we can reduce to an instance without duplicate columns and, in particular, with $n \leq (2\Delta+1)^m$. The running time can be achieved as follows. We create a new matrix for the ILP with all $(2\Delta+1)^m$ possible columns (in lexicographic order) and objective value $c_i = -\infty$ for all columns i. Now we iterate over all n old columns and compute in time $O(m)$ the index of the new column corresponding to the same entries. We then replace its objective value with the current one if this is bigger. In the upcoming running times we will omit the additive term $O(nm)$ and assume the duplicates are already eliminated ($O(\Delta)^m$ is always dominated by actual algorithms running time).

Eisenbrand and Weismantel observed that using the Steinitz Lemma (with ℓ_∞ norm) one can solve integer programs efficiently, if all entries of the matrix are small integers.

Theorem 1 (Steinitz Lemma)**.** *Let $\|\cdot\|$ be a norm in $\mathbb{R}^m$ and $v^{(1)},\dots,v^{(t)} \in \mathbb{R}^m$ such that $\|v^{(i)}\| \leq \Delta$ for all i and $v^{(1)}+\cdots+v^{(t)} = 0$. Then there exists a permutation $\pi \in S_t$ such that for all $j \in \{1,\dots,t\}$*

$$\|\sum_{i=1}^{j} v^{(\pi(i))}\| \leq m\Delta.$$

The proof of the bound $m\Delta$ is due to Sevastyanov [65] (see also [30] for a good overview). Our algorithmic results rely on a similar, but weaker property. Roughly speaking, we only need that there is some $j \approx t/2$ with $\|\sum_{i=1}^{j} v^{(\pi(i))}\|$ being small. All other partial sums are insignificant. As it is a weaker property, we can hope for better bounds than $m\Delta$.

The bounds we need come from discrepancy theory. Let us state some useful definitions and results.

Definition 1. *For a matrix $A \in \mathbb{R}^{m\times n}$ we define its discrepancy as*

$$\operatorname{disc}(A) = \min_{z\in\{0,1\}^n} \left\| A\left(z - \left(\frac{1}{2},\dots,\frac{1}{2}\right)^T\right)\right\|_\infty .$$

Discrepancy theory originates in the problem of coloring the elements of a ground set with two colors such that a given family of subsets are all colored

evenly, i.e., the number of elements of each color is approximately the same. When A is the incidence matrix of this family of sets, z in the definition above gives a coloring and the ℓ_∞ norm its discrepancy. Discrepancy, however, is also studied for arbitrary matrices. If A is the matrix of a linear program as in our case, this definition corresponds to finding an integral solution that approximates $x = (1/2, \dots, 1/2)^T$. Our algorithm is based on dividing a solution into two similar parts. Therefore, discrepancy is a natural measure. However, we need a definition that is stable under restricting to a subset of the columns.

Definition 2. *We define the hereditary discrepancy of a matrix $A \in \mathbb{R}^{m\times n}$ as*

$$\operatorname{herdisc}(A) = \max_{I\subseteq\{1,\dots,n\}} \operatorname{disc}(A_I),$$

where A_I denotes the matrix A restricted to the columns I.

In the following lemma, we will pay a factor of 2 in the discrepancy in order to get a balanced split of the ℓ_1 norm of the solutions.

Lemma 2. *Let $x \in \mathbb{Z}^n_{\geq 0}$. Then there exists a vector $z \in \mathbb{Z}^n_{\geq 0}$ with $z_i \leq x_i$ for all i and*

$$\left\|A(z - \frac{x}{2})\right\|_\infty \leq \operatorname{herdisc}(A).$$

Furthermore, if $\|x\|_1 > 1$, then there exists a vector $z' \in \mathbb{Z}^n_{\geq 0}$ with $z'_i \leq x_i$ for all i, $\frac{1}{6}\cdot\|x\|_1 \leq \|z'\|_1 \leq \frac{5}{6}\cdot\|x\|_1$, and

$$\left\|A(z' - \frac{x}{2})\right\|_\infty \leq 2\cdot\operatorname{herdisc}(A).$$

We remark the symmetry, i.e., when the lemma holds for z (z'), then the same properties hold when substituting z for $x - z$ (z' for $x - z'$) as well.

Proof. Let $x'_i = \lfloor x_i/2 \rfloor$ and $x'' = \lceil x_1/2 \rceil - \lfloor x_i/2 \rfloor \in \{0,1\}$ for all i. Clearly, $x_i = \lfloor x_i/2 \rfloor + \lceil x_i/2 \rceil = 2x'_i + x''_i$. Now apply the definition of $\operatorname{disc}(A_I)$ to x'', where $I = \operatorname{supp}(x'')$ are the the indices i with $x''_i = 1$. This way we obtain a vector $z'' \in \{0,1\}^n$ with $\|A(z'' - x''/2)\|_\infty \leq \operatorname{disc}(A_I) \leq \operatorname{herdisc}(A)$. We now use $z = x' + z''$ to show the first part of the lemma. Then

$$\begin{aligned}\left\|A\left(z - \frac{x}{2}\right)\right\|_\infty &= \left\|A\left((x' + z'') - \frac{2x' + x''}{2}\right)\right\|_\infty \\ &= \left\|A\left(z'' - \frac{x''}{2}\right)\right\|_\infty \leq \operatorname{herdisc}(A).\end{aligned}$$

Furthermore, for all i

$$0 \leq \underbrace{x'_i + z''_i}_{=z_i} \leq x'_i + x''_i \leq 2x'_i + x''_i = x_i.$$

In order to control the ℓ_1 norm in the second part of the lemma, we first split x into two non-empty $y', y'' \in \mathbb{Z}^n_{\geq 0}$ with $y' + y'' = x$ and $\lfloor \|x\|_1/2 \rfloor = \|y'\|_1 \leq$

$\|y''\|_1 = \lceil \|x\|_1 / 2 \rceil$. Now apply the first part of the lemma to obtain $z' \leq y'$ and $z'' \leq y''$ with $\|A(z' - y'/2)\|_\infty \leq \operatorname{herdisc}(A)$ and $\|A(z'' - y''/2)\|_\infty \leq \operatorname{herdisc}(A)$. We can assume w.l.o.g. that $\|z'\|_1 \leq \|y'\|_1 / 2 \leq \|x\|_1 / 4$ and $\|z''\|_1 \geq \|y''\|_1 / 2 \geq \|x\|_1 / 4$, since otherwise we can swap them for $y' - z'$ and $y'' - z''$, respectively. We will use $z = z' + z''$ for the second part of the lemma. As for the lower bound,

$$\|z\|_1 \geq \|z''\|_1 \geq \frac{\|x\|_1}{4}.$$

For the upper bound we first consider the case where $\|x\|_1 \leq 5$ and note that $\|z'\|_1 \leq \|y'\|_1 / 2 = \|y'\|_1 - \|y'\|_1 / 2 < \|y'\|_1$. Thus,

$$\|z\|_1 = \|z' + z''\|_1 \leq \|y' + y''\|_1 - 1 \leq \|x\|_1 - \frac{1}{5}\|x\|_1 = \frac{4}{5}\|x\|_1.$$

If $\|x\|_1 \geq 6$,

$$\begin{aligned}\|z\|_1 = \|z' + z''\|_1 &\leq \frac{\|x\|_1}{4} + \|y''\|_1 = \frac{\|x\|_1}{4} + \left\lceil \frac{\|x\|_1}{2} \right\rceil \\ &\leq \frac{\|x\|_1}{2} + \frac{\|x\|_1}{4} + \frac{1}{2} \leq \frac{3}{4}\|x\|_1 + \frac{1}{12}\|x\|_1 \leq \frac{5}{6}\|x\|_1.\end{aligned}$$

Finally, $z_i = z_i' + z_i'' \leq y_i' + y_i'' = x_i$ and

$$\begin{aligned}\left\|A\left(z - \frac{x}{2}\right)\right\|_\infty &= \left\|A\left((z' + z'') - \frac{y' + y''}{2}\right)\right\|_\infty \\ &\leq \left\|A\left(z' - \frac{y'}{2}\right)\right\|_\infty + \left\|A\left(z'' - \frac{y''}{2}\right)\right\|_\infty \leq 2 \cdot \operatorname{herdisc}(A). \quad \square\end{aligned}$$

Since our algorithm's running time will depend on $\operatorname{herdisc}(A)$, it will be useful to state some bounds.

Theorem 3 (Spencer's Six Standard Deviations Suffice [66])**.** *For every matrix A with biggest absolute value of an entry Δ,*

$$\operatorname{herdisc}(A) \leq 6\sqrt{m} \cdot \Delta.$$

There are matrices for which this bound is tight up to a constant factor. For specific matrices it might be lower. The linear dependency on Δ, however, is required for any matrix A.

Lemma 4. *For every matrix $A \in \mathbb{R}^{m \times n}$ with absolute value of an entry $\leq \Delta$,*

$$\operatorname{herdisc}(A) \geq \frac{\Delta}{2}.$$

This can be seen by taking $I = \{i\}$ in the definition of $\operatorname{herdisc}(A)$ with A_i being a column with an entry of absolute value Δ. The dependency on m can be lower for specific matrices. For example, matrices with a small ℓ_1 norms in every column yield better bounds.

Theorem 5 (Beck, Fiala [13]). *For every matrix $A \in \mathbb{R}^{m\times n}$, where the ℓ_1 norm of each column is at most t,*

$$\operatorname{herdisc}(A) < t.$$

Moving back to pseudo-polynomial integer programming, the first algorithm by Papadimitriou relies on the following bound on the ℓ_1 norm of an optimal solution.

Lemma 6 (Papadimitriou [61]). *Let $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ be bounded and feasible. Then there exists an optimal solution x^* with*

$$\|x^*\|_1 \leq n^2(m(\Delta + \|b\|_\infty))^{2m+1} \leq (m(\Delta + \|b\|_\infty))^{O(m)}$$

In other words, $\|x^*\|_1 \leq 2^K$, where $K = O(m\log(m) + m\log(\Delta + \|b\|_\infty))$.

We also need this for our algorithm. A similar bound could also be obtained via the Steinitz Lemma.

2.2 DYNAMIC PROGRAM

In this section we will show how to compute the best solution x^* to an ILP with the additional constraint $\|x^*\|_1 \leq (6/5)^K$. If the ILP is bounded, then with $K = O(m\log(m) + m\log(\Delta + \|b\|_\infty))$ this is indeed the optimum to the ILP (Lemma 6). In Section 2.2 we discuss how to cope with unbounded ILPs.

Let $H \geq \operatorname{herdisc}(A)$ be an upper bound on the hereditary discrepancy. For every $i = 0, 1 \ldots, K$ and every b' with $\|b' - 2^{i-K} \cdot b\|_\infty \leq 4H$ we solve

$$\max\left\{c^T x : Ax = b', \|x\|_1 \leq \left(\frac{6}{5}\right)^i, x \in \mathbb{Z}^n_{\geq 0}\right\}. \tag{1}$$

We iteratively derive solutions for i using pairs of solutions for $i-1$. Ultimately, we will compute a solution for $i = K$ and $b' = b$.

If $i = 0$, then the solutions are trivial, since $\|x\|_1 \leq 1$. This means they correspond exactly to the columns of A. Fix some $i > 0$ and b' and let x^* be an optimal solution to (1). By Lemma 2 there exists a $0 \leq z \leq x^*$ with $\|Az - b'/2\|_\infty \leq 2 \cdot \operatorname{herdisc}(A)$ and

$$\|z\|_1 \leq \frac{5}{6}\|x^*\|_1 \leq \frac{5}{6} \cdot \left(\frac{6}{5}\right)^i = \left(\frac{6}{5}\right)^{i-1},$$

if $\|x^*\|_1 > 1$, or $\|z\|_1 \leq \|x^*\|_1 \leq 1 \leq (6/5)^{i-1}$, otherwise. The same holds for $x^* - z$. Therefore, z is an optimal solution to $\max\{c^T x : Ax = b'', \|x\|_1 \leq (6/5)^{i-1}, x \in \mathbb{Z}^n_{\geq 0}\}$ where $b'' = Az$. Likewise, $x^* - z$ is an optimal solution to $\max\{c^T x : Ax = b' - b'', \|x\|_1 \leq (6/5)^{i-1}, x \in \mathbb{Z}^n_{\geq 0}\}$. We claim that $\|b'' - 2^{(i-1)-K} \cdot b\|_\infty \leq 4H$ and $\|(b' - b'') - 2^{(i-1)-K} \cdot b\|_\infty \leq 4H$. This implies that we can look up solutions for b'' and $b' - b''$ in the dynamic table and their sum is a solution for b'. Clearly it is also optimal. We do not know b'', but we can guess it: There are only $(8H+1)^m$ candidates. To compute an entry, we therefore enumerate all possible b'' and take the two partial solutions (for b'' and $b' - b''$), where the sum of both values is maximized.

PROOF OF CLAIM. We have that

$$\begin{aligned}\left\|b'' - 2^{(i-1)-K}b\right\|_\infty &= \left\|Az - \frac{1}{2}b' + \frac{1}{2}b' - 2^{(i-1)-K}b\right\|_\infty \\ &\leq \left\|Az - \frac{1}{2}b'\right\|_\infty + \left\|\frac{1}{2}b' - 2^{(i-1)-K}b\right\|_\infty \\ &\leq 2 \cdot \text{herdisc}(A) + \frac{1}{2}\left\|b' - 2^{i-K}b\right\|_\infty \leq 4H.\end{aligned}$$

The same holds for $b' - b''$, since z and $x^* - z$ are interchangeable.

Naive running time

Note that in our use of the O-notation we can hide factors polynomial in m: $O(H)^{2m} = O(H)^{2m} \cdot 2^m$. The dynamic table has $(K+1) \cdot O(H)^m$ entries. To compute an entry, $O(n \cdot m) \leq O(\Delta)^m \leq O(H)^m$ operations are necessary during initialization and $O(H)^m$ in the iterative calculations. This gives a total running time of

$$\begin{aligned}O(H)^{2m} \cdot (K+1) &= O(H)^{2m} \cdot (m\log(m) + m\log(\Delta + \|b\|_\infty)) \\ &= O(H)^{2m} \cdot (\log(\Delta + \|b\|_\infty)).\end{aligned}$$

Unbounded solutions

In the previous dynamic program there is no mechanism for detecting when the ILP is unbounded. To handle unbounded ILPs we follow the approach from [30]. The ILP $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ is unbounded, if and only if $\{x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ has a solution and $\max\{c^T x : Ax = 0, x \in \mathbb{Z}^n_{\geq 0}\}$ has a solution with positive objective value. After running the dynamic program - thereby verifying that there exists any solution - we have to check if the latter condition holds. We can simply run the algorithm again on $\max\{c^T x : Ax = 0, x \in \mathbb{Z}^n_{\geq 0}\}$ with $K = O(m\log(m) + m\log(\Delta))$. If it returns a positive value, the ILP is unbounded. Let us argue why this is enough. Suppose that $\max\{c^T x : Ax = 0, x \in \mathbb{Z}^n_{\geq 0}\}$ has a solution x^* with a positive objective value. Let $N = \|x^*\|_1$. The ILP $\max\{c^T x : Ax = 0, \|x\|_1 \leq N, x \in \mathbb{Z}^n_{\geq 0}\}$ is clearly feasible and bounded. By Lemma 6 it has an optimal solution (with positive objective value) with ℓ_1 norm at most 2^K, where $K = O(m\log(m) + m\log(\Delta))$. Hence, running the algorithm with this choice of K suffices to check if there is a positive solution.

2.3 IMPROVEMENTS TO THE RUNNING TIME

Applying convolution

Can we speed up the computation of entries in the dynamic table? Let D_i be the set of vectors b' with $\|b' - 2^{i-K} \cdot b\|_\infty \leq 4H$. Recall, the dynamic programs computes values for each element in $D_0, D_1, \ldots, D_K$. More precisely,

for the value of $b' \in D_i$ we consider vectors b'' such that $b'', b' - b'' \in D_{i-1}$ and take the maximum sum of the values for $b'', b' - b''$ among all. First consider only the case of $m = 1$. Here we have that $b' \in D_i$ is equivalent to $-4H \leq b' - 2^{i-K} \cdot b \leq 4H$. This problem is well studied. It is a variant of (min, +)-convolution.

(MIN, +)-CONVOLUTION

INPUT: $r_1, \ldots, r_n$ and $s_1, \ldots, s_n$.

OUTPUT: $t_1, \ldots, t_n$, where $t_k = \min_{i+j=k} r_i + s_j$.

(max, +)-convolution is the counterpart where the maximum is taken instead of the minimum. The two problems are equivalent. Each of them can be transformed to the other by negating the elements. (min, +)-convolution admits a trivial $O(n^2)$ time algorithm and it has been conjectured that there exists no truly sub-quadratic algorithm [25]. There does, however, exist an $O(n^2/\log(n))$ time algorithm [16], which we are going to use. In fact, there is an even faster algorithm that runs in $O(n^2/2^{\Omega(\sqrt{\log(n)})})$ [21].

We will now create an instance of (max, +)-convolution for calculating D_i from D_{i-1}. We first deal with the problem that $2^{i-1-k}b$ might not be integral. Let $b^0 = \lfloor 2^{i-1-k}b \rfloor$ denote the vector rounded down in every component. D_{i-1} is completely covered by the points with distance $4H + 2$ from b^0. Likewise, D_i is covered by the points with distance $4H + 2$ from $2b^0$.

We project a vector $b' \in D_{i-1}$ to

$$f_{i-1}(b') = \sum_{j=1}^{m} (16H + 11)^{j-1} \underbrace{(4H + 3 + b'_j - b^0)}_{\in \{1,\ldots,8H+5\}}. \tag{2}$$

The value $16H + 11$ is chosen, because it bigger than the sum of two values of the form $4H + 3 + b'_j - b^0$. We define $f_i(b')$ for all $b' \in D_i$ in the same way, except we substitute b^0 for $2b^0$. For all $a, a' \in D_{i-1}, b' \in D_i$, it holds that $f_{i-1}(a) + f_{i-1}(a') = f_i(b')$, if and only if $a + a' = b' - (4H + 3, \ldots, 4H + 3)^T$:

Proof $\Rightarrow$. Let $f_{i-1}(a) + f_{i-1}(a') = f_i(b')$. Then, in particular,

$$f_{i-1}(a) + f_{i-1}(a') \equiv f_i(b') \mod 16H + 11$$

Since all but the first element of the sum (2) are multiples of $16H + 11$, i.e., they are equal 0 modulo $16H + 11$, we can omit them in the equation. Hence,

$$(4H + 3 + a_1 - b_1^0) + (4H + 3 + a'_1 - b_1^0) \equiv (4H + 3 + b'_1 - 2b_1^0) \mod 16H + 11.$$

We even have equality (without modulo) here, because both sides are smaller than $16m\Delta + 11$. Simplifying the equation gives $a_1 + a'_1 = b'_1 - (4H + 3)$. Now consider again the equation $f_{i-1}(a) + f_{i-1}(a') = f_i(b')$. In the sums leave out the first element. The equation still holds, since by the elaboration

above this changes the left and right hand-side by the same value. We can now repeat the same argument to obtain $a_2 + a'_2 = b'_2 - (4H+3)$ and the same for all other dimensions. □

Proof $\Leftarrow$. Let $a + a' = b' - (4H+3, \ldots, 4H+3)^T$. Then for every j,

$$(4H + 3 + a_j - b^0_j) + (4H + 3 + a'_j - b^0_j) = 4H + 3 + b'_j - 2b^0_j.$$

It directly follows that $f_{i-1}(a) + f_{i-1}(a') = f_i(b')$. □

This means when we write the value of each $b'' \in D_{i-1}$ to r_j and s_j, where $j = f_{i-1}(b'')$ and every entry not used is set to $-\infty$, the correct solutions will be in t. More precisely, we can read the result for some $b' \in D_i$ at t_j where $j = f_i(b' + (4H+3, \ldots, 4H+3)^T)$.

With an algorithm for (min, +)-convolution with running time $T(n)$ we get an algorithm with running time $T(O(H)^m) \cdot (m \log(m) + m \log(\Delta + \|b\|_\infty))$. Inserting $T(n) = n^2 / \log(n)$ and using $H \geq \Delta/2$ we get:

Theorem 7. *There exists an algorithm that finds the optimum of* $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$*, in time* $O(H)^{2m} \cdot \log(\Delta + \|b\|_\infty) / \log(\Delta)$.

Clearly, a sub-quadratic algorithm, where $T(n) = n^{2-\delta}$ for some $\delta > 0$, would directly improve the exponent. Next, we will consider the problem of only testing feasibility of an ILP. Since we only record whether or not there exists a solution for a particular right-hand side, the convolution problem reduces to the following.

BOOLEAN CONVOLUTION
INPUT: $r_1, \ldots, r_n \in \{0,1\}$ and $s_1, \ldots, s_n \in \{0,1\}$.
OUTPUT: $t_1, \ldots, t_n \in \{0,1\}$, where $t_k = \bigvee_{i+j=k} r_i \wedge s_j$.

This problem can be solved very efficiently via fast Fourier transform. We compute the $(+,\cdot)$-convolution of the input. It is well known that this can be done using FFT in time $O(n \log(n))$. The $(+,\cdot)$-convolution of r and s is the vector t, where $t_k = \sum_{i+j=k} r_i \cdot s_j$. To get the Boolean convolution instead, we simply replace each $t_k > 0$ by 1. Using $T(n) = O(n \log(n))$ for the convolution algorithm we obtain the following.

Theorem 8. *There exists an algorithm that finds an element in* $\{x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$*, if there is one, in time* $O(H)^m \cdot \log(\Delta) \cdot \log(\Delta + \|b\|_\infty)$.

This can be seen from the calculation below. Note that we can scrape off factors polynomial in m and $\log(H) \leq O(m \log(\Delta))$:

$$\begin{aligned} O(H)^m \cdot m \log(H) \cdot (m \log(m) + m \log(\Delta + \|b\|_\infty)) \\ \leq O(H)^m \cdot \log(\Delta) \cdot \log(\Delta + \|b\|_\infty)) \end{aligned}$$

Use of proximity

Eisenbrand and Weismantel gave the following bound on the proximity between continuous and integral solutions.

Theorem 9 ([30]). *Let* $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ *be feasible and bounded. Let* x^* *be an optimal vertex solution of the fractional relaxation. Then there exists an optimal solution* z^* *with*

$$\|z^* - x^*\|_1 \leq m(2m\Delta + 1)^m.$$

We briefly explain, how they use this theorem to reduce the right-hand side b at the expense of computing the optimum of the fractional relaxation: Note that $z^*_i \geq \ell_i := \max\{0, \lceil x^*_i \rceil - m(2m\Delta + 1)^m\}$. Since x^* is a vertex solution, it has at most m non-zero components. By setting $y = x - \ell$ we obtain the equivalent ILP $\max\{c^T y : Ay = b - A\ell, y \in \mathbb{Z}^n_{\geq 0}\}$. Indeed, this ILP has a bounded right-hand side:

$$\|b - A\ell\|_\infty = \|A(x^* - \ell)\|_\infty \leq \Delta m^2 (2m\Delta + 1)^m = O(m\Delta)^{m+1}.$$

Here, we use that x^* and ℓ differ only in non-zero components of x^* and in those by at most $m(2m\Delta + 1)^m$. Like in earlier bounds, the O-notation hides polynomial terms in m. Using the $n \cdot O(m\Delta)^{2m} \cdot \|b\|_1^2$ time algorithm from [30], this gives a running time of $n \cdot O(m\Delta)^{4m+2} + \mathrm{LP}$, where LP is the time to solve the relaxation. The logarithmic dependence on $\|b\|_\infty$ in our new algorithm leads to a much smaller exponent: Using Theorem 7 and the construction above, the ILP can be solved in time $O(H)^{2m} + \mathrm{LP}$. Feasibility can be tested in time $O(H)^m \cdot \log^2(\Delta) + \mathrm{LP}$ using Theorem 8.

Special forms of matrices

Let $\Delta_1, \ldots, \Delta_m \leq \Delta$ denote the largest absolute values of each row in A. When some of these values are much smaller than Δ, the maximum among all, we can do better than $O(\sqrt{m}\Delta)^{2m} \cdot \log(\|b\|_\infty)$. An example for a heterogeneous matrix is Unbounded Knapsack with cardinality constraints. More generally, consider some scalars $a_1, \ldots, a_m > 0$ that we pre-multiply each row with, i.e., define $A' = \mathrm{diag}(a_1, \ldots, a_m) \cdot A$, where

$$\mathrm{diag}(a_1, \ldots, a_m) = \begin{pmatrix} a_1 & & 0 \\ & \ddots & \\ 0 & & a_m \end{pmatrix}.$$

We claim that in the dynamic program a table of size $\prod_{k=1}^m O(H'/a_k)$ suffices, where $H' \geq \mathrm{herdisc}(A')$. With $a_k = 1/\Delta_k$ in the setting above, this gives $\prod_{k=1}^m O(\sqrt{m}\Delta_k)$. Clearly, the ILP $\max\{c^T x, Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ is equivalent to

$$\max\{c^T x, A'x = b, x \in \mathbb{Z}^n_{\geq 0}\},$$

where $b' = \operatorname{diag}(a_1, \ldots, a_m) \cdot b$. At first glance, our algorithm cannot be applied to this problem, since the entries are not integral. However, in the algorithm we only use the fact that the number of points Ax with $x \in \mathbb{Z}^n_{\geq 0}$ close to some point b'', i.e., with $\|Ax - b''\|_\infty \leq 4H$, is small and can be enumerated. The points $A'x$ with $x \in \mathbb{Z}^n_{\geq 0}$ and $\|A'x - b''\|_\infty \leq 4H'$ are exactly those with $|(Ax)_k - b''_k/a_k| \leq 4H'/a_k$ for all k. These are $\prod_{k=1}^m O(H'/a_k)$ many and they can be enumerated. This way, we get a running time of

$$\prod_{k=1}^{m} O(H'/a_k)^2 \cdot \log(\Delta + \|b\|_\infty).$$

When the objective function has small, integral coefficients, it can be more efficient to perform a binary search for the optimum and encode the objective function as an additional constraint. We can bound the optimum by $(m(\Delta + \|b\|_\infty))^{O(m)} \cdot \|c\|_\infty$ using the bound on the ℓ_1 norm of the solution. Hence, the binary search takes at most $O(m\log(m) + m\log(\Delta + \|b\|_\infty) + \log(\|c\|_\infty)) \leq O(m\log(m + \Delta + \|c\|_\infty + \|b\|_\infty))$ iterations. For a guess τ the following feasibility ILP tests if there is a solution of value at least τ.

$$\begin{pmatrix} c_1 & \ldots & c_n & -1 \\ & & & 0 \\ & A & & \vdots \\ & & & 0 \end{pmatrix} x = \begin{pmatrix} \tau \\ b_1 \\ \vdots \\ b_n \end{pmatrix}$$
$$x \in \mathbb{Z}^{n+1}_{\geq 0}$$

We can solve the ILP above in time

$$\begin{aligned} &T(\sqrt{m+1}\|c\|_\infty \cdot \prod_{k=1}^{m} O(\sqrt{m+1}\Delta_k)) \cdot \log(\|b\|_\infty + \tau) \\ &\qquad \leq T(\|c\|_\infty \cdot \prod_{k=1}^{m} O(\sqrt{m}\Delta_k)) \cdot m\log(m + \Delta + \|c\|_\infty + \|b\|_\infty), \end{aligned}$$

where $T(n) = O(n\log(n))$ is the running time of Boolean convolution. By adding the time for the binary search and by hiding polynomials in m, we get the total running time of

$$\|c\|_\infty \cdot \prod_{k=1}^{m} [O(\sqrt{m}\Delta_k)] \cdot \log(\Delta + \|c\|_\infty) \cdot \log^2(\Delta + \|c\|_\infty + \|b\|_\infty).$$

2.4 LOWER BOUNDS

Optimization problem

We use an equivalence between the problems UNBOUNDED KNAPSACK and (min, +)-convolution regarding sub-quadratic algorithms.

UNBOUNDED KNAPSACK

INPUT: $C \in \mathbb{N}$, $w_1, \ldots, w_n \in \mathbb{N}$, and $p_1, \ldots, p_n \in \mathbb{N}$.

OUTPUT: Multiplicities $x_1, \ldots, x_n$, such that $\sum_{i=1}^{n} x_i \cdot w_i \leq C$ and $\sum_{i=1}^{n} x_i \cdot p_i$ is maximized.

Note that when we instead require $\sum_{i=1}^{n} x_i \cdot w_i = C$ in the problem above, we can transform it to this form by adding an item of profit zero and weight 1.

Theorem 10 ([25]). *For any $\delta > 0$ there exists no $O((n+C)^{2-\delta})$ time algorithm for UNBOUNDED KNAPSACK unless there exists a truly sub-quadratic algorithm for (min, +)-convolution.*

When using this theorem, we assume that the input already consists of the at most C relevant items only, $n \leq C$, and $w_i \leq C$ for all i. This preprocessing can be done in time $O(n+C)$.

Theorem 11. *For every fixed m there does not exist an algorithm that solves ILPs with m constraints in time $f(m) \cdot (n^{2-\delta} + (\Delta + \|b\|_\infty)^{2m-\delta})$ for some $\delta > 0$ and a computable function f, unless there exists a truly sub-quadratic algorithm for (min, +)-convolution.*

Proof. Let $\delta > 0$ and $m \in \mathbb{N}$. Assume that there exists an algorithm that solves ILPs of the form $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ where $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, and $c \in \mathbb{Z}^n$ in time $f(m) \cdot (n^{2-\delta} + (\Delta + \|b\|_\infty)^{2m-\delta})$, where Δ is the greatest absolute value in A. We will show that this implies an $O((n+C)^{2-\delta'})$ time algorithm for the UNBOUNDED KNAPSACK PROBLEM for some $\delta' > 0$. Let $(C, (w_i)_{i=1}^n, (p_i)_{i=1}^n)$ be an instance of this problem. Let us first observe that the claim holds for $m = 1$. Clearly the UNBOUNDED KNAPSACK PROBLEM (with equality) can be written as the following ILP (UKS1).

$$\max \sum_{i=1}^{n} p_i \cdot x_i$$
$$\sum_{i=1}^{n} w_i \cdot x_i = C$$
$$x \in \mathbb{Z}^n_{\geq 0}$$

Since $w_i \leq C$ for all i (otherwise the item can be discarded), we can solve this ILP by assumption in time $f(1) \cdot (n^{2-\delta} + (2C)^{2-\delta}) \leq O((n+C)^{2-\delta})$. Now consider the case where $m > 1$. We want to reduce Δ by exploiting the additional rows. Let $\Delta = \lfloor C^{1/m} \rfloor + 1 > C^{1/m}$. We write C in base-Δ notation, i.e.,

$$C = C^{(0)} + \Delta C^{(1)} + \cdots + \Delta^{m-1} C^{(m-1)},$$

where $0 \le C^{(k)} < \Delta$ for all k. Likewise, write $w_i = w_i^{(0)} + \Delta w_i^{(1)} + \cdots + \Delta^{m-1} w_i^{(m-1)}$ with $0 \le w_i^{(k)} < \Delta$ for all k. We claim that (UKS1) is equivalent to the following ILP (UKSm).

$$\max \sum_{i=1}^{n} p_i \cdot x_i$$

$$\sum_{i=1}^{n} [w_i^{(0)} \cdot x_i] - \Delta \cdot y_1 = C^{(0)} \tag{3}$$

$$\sum_{i=1}^{n} [w_i^{(1)} \cdot x_i] + y_1 - \Delta \cdot y_2 = C^{(1)} \tag{4}$$

$$\vdots$$

$$\sum_{i=1}^{n} [w_i^{(m-2)} \cdot x_i] + y_{m-2} - \Delta \cdot y_{m-1} = C^{(m-2)} \tag{5}$$

$$\sum_{i=1}^{n} [w_i^{(m-1)} \cdot x_i] + y_{m-1} = C^{(m-1)} \tag{6}$$

$$x \in \mathbb{Z}^n_{\ge 0}$$
$$y \in \mathbb{Z}^m_{\ge 0}$$

CLAIM $x \in (\mathrm{USK1}) \Rightarrow x \in (\mathrm{USKm})$. Let x be a solution to (UKS1). Then for all $1 \le \ell \le m$,

$$\sum_{i=1}^{n} \sum_{k=0}^{\ell-1} \Delta^k w_i^{(k)} \cdot x_i \equiv \sum_{i=1}^{n} w_i \cdot x_i \equiv C \equiv \sum_{k=0}^{\ell-1} \Delta^k C^{(k)} \mod \Delta^\ell.$$

This is because all $\Delta^\ell w_i^{(\ell)}, \ldots, \Delta^{m-1} w_i^{(m-1)}$ and $\Delta^\ell C^{(\ell)}, \ldots, \Delta^{m-1} C^{(m-1)}$ are multiples of Δ^ℓ. It follows that there exists an $y_\ell \in \mathbb{Z}$ such that

$$\sum_{i=1}^{n} \sum_{k=0}^{\ell-1} [\Delta^k w_i^{(k)} \cdot x_i] - \Delta^\ell \cdot y_\ell = \sum_{k=0}^{\ell-1} \Delta^k C^{(k)}.$$

Furthermore, y_ℓ is non-negative, because otherwise

$$\begin{aligned} \sum_{k=0}^{\ell-1} \Delta^k C^{(k)} &\le \sum_{k=0}^{\ell-1} \Delta^k (\Delta - 1) < \Delta^{\ell-1} (\Delta - 1) \sum_{k=0}^{\infty} \Delta^{-k} \\ &= \Delta^{\ell-1} \frac{\Delta - 1}{1 - \frac{1}{\Delta}} = \Delta^\ell \le -\Delta^\ell y_\ell \le \sum_{i=1}^{n} \sum_{k=0}^{\ell-1} [\Delta^k w_i^{(k)} \cdot x_i] - \Delta^\ell y_\ell. \end{aligned}$$

We choose $y_1,\dots,y_m$ exactly like this. The first constraint (3) follows directly. Now let $\ell \in \{2,\dots,m\}$. By choice of $y_{\ell-1}$ and y_ℓ we have that

$$\sum_{i=1}^{n}\Bigg[\underbrace{\left(\sum_{k=0}^{\ell-1}\Delta^k w_i^{(k)} - \sum_{k=0}^{\ell-2}\Delta^k w_i^{(k)}\right)}_{=\Delta^{\ell-1}w_i^{(\ell-1)}}\cdot x_i\Bigg] + \Delta^{\ell-1}\cdot y_{\ell-1} - \Delta^{\ell}\cdot y_\ell$$
$$= \underbrace{\sum_{k=0}^{\ell-1}\Delta^k C^{(k)} - \sum_{k=0}^{\ell-2}\Delta^k C^{(k)}}_{=\Delta^{\ell-1}C^{(\ell-1)}}. \quad (7)$$

Dividing both sides by $\Delta^{\ell-1}$ we get every constraint (4) - (5) for the correct choice of ℓ. Finally, consider the special case of the last constraint (6). By choice of y_m we have that

$$\sum_{i=1}^{n}\underbrace{\sum_{k=0}^{m-1}\Delta^k w_i^{(k)}}_{=w_i}\cdot x_i - \Delta^m\cdot y_m = \underbrace{\sum_{k=0}^{m-1}\Delta^k C^{(k)}}_{=C}.$$

Thus, $y_m = 0$ and (7) implies the last constraint (with $\ell = m$).

CLAIM $x \in (\text{USKm}) \Rightarrow x \in (\text{USK1})$. Let $x_1,\dots,x_n,y_1,\dots,y_{m-1}$ be a solution to (UKSm) and set $y_m = 0$. We show by induction that for all $\ell \in \{1,\dots,m\}$

$$\sum_{i=1}^{n}\sum_{k=0}^{\ell-1}\Delta^k w_i^{(k)}\cdot x_i - \Delta^\ell y_\ell = \sum_{k=0}^{\ell-1}\Delta^k C^{(k)}.$$

With $\ell = m$ this implies the claim as $y_m = 0$ by definition. For $\ell = 1$ the equation is exactly the first constraint (3). Now let $\ell > 1$ and assume that the equation above holds. We will show that it also holds for $\ell+1$. From (USKm) we have

$$\sum_{i=1}^{n}[w_i^{(\ell)}\cdot x_i] + y_\ell - \Delta\cdot y_{\ell+1} = C^{(\ell)}.$$

Multiplying each side by Δ^ℓ we get

$$\sum_{i=1}^{n}[\Delta^\ell w_i^{(\ell)}\cdot x_i] + \Delta^\ell y_\ell - \Delta^{\ell+1}\cdot y_{\ell+1} = \Delta^\ell C^{(\ell)}.$$

By adding and subtracting the same elements, it follows that

$$\sum_{i=1}^{n}\left[\left(\sum_{k=0}^{\ell}\Delta^k w_i^{(k)} - \sum_{k=0}^{\ell-1}\Delta^k w_i^{(k)}\right)\cdot x_i\right] + \Delta^\ell\cdot y_\ell - \Delta^{\ell+1}\cdot y_{\ell+1}$$
$$= \sum_{k=0}^{\ell}\Delta^k C^{(k)} - \sum_{k=0}^{\ell-1}\Delta^k C^{(k)}.$$

By inserting the induction hypothesis we conclude

$$\sum_{i=1}^{n}\sum_{k=0}^{\ell}[\Delta^k w_i^{(k)}\cdot x_i]-\Delta^{\ell+1}y_{\ell+1}=\sum_{k=0}^{\ell}\Delta^k C^{(k)}.$$

CONSTRUCTING AND SOLVING THE ILP. The ILP (UKSm) can be constructed easily in $O(Cm+nm)\le O((n+C)^{2-\delta/m})$ operations (recall that m is a constant). We obtain $\Delta=\lfloor C^{1/m}\rfloor+1$ by guessing: More precisely, we iterate over all numbers $\Delta_0\le C$ and find the one where $(\Delta_0-1)^m<C\le\Delta_0^m$. There are of course more efficient, non-trivial ways to compute the rounded m-th root. The base-Δ representation for $w_1,\dots,w_n$ and C can be computed with $O(m)$ operations for each of these numbers.

All entries of the matrix in (UKSm) and the right-hand side are bounded by $\Delta=O(C^{1/m})$. Therefore, by assumption this ILP can be solved in time

$$f(m)\cdot(n^{2-\delta}+O(C^{1/m})^{2m-\delta})\le f(m)\cdot O(1)^{2m-\delta}\cdot(n+C)^{2-\delta/m}=O((n+C)^{2-\delta/m}).$$

This would yield a truly sub-quadratic algorithm for the UNBOUNDED KNAPSACK PROBLEM. □

Feasibility problem

We will show that our algorithm for solving feasibility of ILPs is optimal (except for log factors). We use a recently discovered lower bound for k-SUM based on the SETH.

k-SUM
INPUT: $T\in\mathbb{N}_0$ and $Z_1,\dots,Z_k\subset\mathbb{N}_0$ where $\lvert Z_1\rvert+\lvert Z_2\rvert+\cdots+\lvert Z_k\rvert=n\in\mathbb{N}$.
OUTPUT: $z_1\in Z_1, z_2\in Z_2,\dots,z_k\in Z_k$ such that $z_1+z_2+\cdots+z_k=T$.

Theorem 12 ([1]). *If the SETH holds, then for every $\delta>0$ there exists a value $\gamma>0$ such that k-SUM cannot be solved in time $O(T^{1-\delta}\cdot n^{\gamma k})$.*

This implies that for every $p\in\mathbb{N}$ there is no $O(T^{1-\delta}\cdot n^p)$ time algorithm for k-SUM if $k\ge p/\gamma$.

Theorem 13. *If the SETH holds, for every fixed m there does not exist an algorithm that solves feasibility of ILPs with m constraints in time $n^{f(m)}\cdot(\Delta+\|b\|_\infty)^{m-\delta}$.*

Proof. Like in the previous reduction we start with the case of $m=1$. For higher values of m the result can be shown in the same way as before.

Suppose there exists an algorithm for solving feasibility of ILPs with one constraint in time $n^{f(1)}\cdot(\Delta+\|b\|_\infty)^{1-\delta}$ for some $\delta>0$ and $f(1)\in\mathbb{N}$. Set $k=\lceil f(1)/\gamma\rceil$ with γ as in in Theorem 12 and consider an instance $(T,Z_1,\dots,Z_k)$

of k-SUM. We will show that this can be solved in $O(T^{1-\delta} \cdot n^{f(1)})$, which contradicts the SETH. For every $i \leq k$ and every $z \in Z_i$ we use a binary variable $x_{i,z}$ that describes whether z is used. We can easily model k-SUM as the following ILP:

$$\sum_{i=1}^{k} \sum_{z \in Z_i} z \cdot x_{i,z} = T$$
$$\sum_{z \in Z_i} x_{i,z} = 1 \qquad \forall i \in \{1, \ldots, k\}$$
$$x_{i,z} \in \mathbb{Z}_{\geq 0} \qquad \forall i \in \{1, \ldots, k\}, z \in Z_i$$

However, since we want to reduce to an ILP with one constraint, we need a slightly more sophisticated construction. We will show that the cardinality constraints can be encoded into the k-SUM instance by increasing the numbers by a factor of $2^{O(k)}$, which is in $O(1)$ since k is some constant depending on $f(1)$ and γ only. We will use this to obtain an ILP with only one constraint and values of size at most $O(T)$. A similar construction is also used in [1].

Our goal is to construct an instance $(T', Z'_k, \ldots, Z'_k)$ such that for every x^* it holds that x^* is a solution to the first ILP if and only if $x^* \in \{x : \sum_{i=1}^{k} \sum_{z \in Z'_i} z \cdot x_{i,z} = T', x \in \mathbb{Z}^n_{\geq 0}\}$ $(*)$. We will use one element to represent each element in the original instance. Consider the binary representation of numbers in $Z'_1 \cup \cdots \cup Z'_k$ and of T'. The numbers in the new instance will consist of three parts and $\lceil \log(k) \rceil$ many os between them to prevent interference. For an illustration of the construction see Figure 3. The $\lceil \log(k) \rceil$ most significant bits ensure that exactly k elements are selected; the middle part are k bits that ensure of every set Z'_i exactly one element is selected; the least significant $\lceil \log(T) \rceil$ bits represent the original values of the elements. Set the values in the first part of the numbers to 1 for all elements $Z'_1 \cup \cdots \cup Z'_k$ and to k in T'. Clearly this ensures that at most k elements are chosen. The sum of at most k elements cannot be larger than $k \leq 2^{\lceil \log(k) \rceil}$ times the biggest element. This implies that the buffers of $\lceil \log(k) \rceil$ zeroes cannot overflow and we can consider each of the three parts independently. It follows that exactly k elements must be chosen by any feasible solution. The system $\{x : \sum_{i=1}^{k} 2^i x_i = 2^{k+1} - 1, \|x\|_1 = k, \mathbb{Z}^k_{\geq 0}\}$ has exactly one solution and this solution is $(1, 1, \ldots, 1)$: Consider summing up k powers of 2 and envision the binary representation of the partial sums. When we add some 2^i to the partial sum, the number of ones in the binary representation increases by one, if the i'th bit of the current sum is zero. Otherwise, it does not increase. However, since in the binary representation of the final sum there are k ones, it has to increase in each addition. This means no power of two can be added twice and therefore each has to be added exactly once.

It follows that the second part of the numbers enforces that of every Z'_i exactly one element is chosen. We conclude that $(*)$ solves the initial k-SUM instance. By assumption this can be done in time $n^{f(1)} \cdot (\Delta + \|b\|_\infty)^{1-\delta} = n^{f(1)} \cdot O(T')^{1-\delta} = O(n^{f(1)} \cdot T^{1-\delta})$. Here we use that $T' \leq 2^{3\log(k)+k+\log(T)+4} = O(k^3 2^k T) = O(T)$, since k is a constant.

$$Z_i' \ni z' = \overbrace{\underbrace{0\dots0001}_{\lceil\log(k)\rceil}}^{\text{bin}(1)} | \underbrace{0\dots0}_{\lceil\log(k)\rceil} | \overbrace{\underbrace{0\dots010\dots0}_{k}}^{\text{bin}(2^i)} | \underbrace{0\dots0}_{\lceil\log(k)\rceil} | \overbrace{\underbrace{0110\dots}_{\lceil\log(T)\rceil}}^{\text{bin}(z)}$$

$$T' = \overbrace{\underbrace{0\dots1011}_{\lceil\log(k)\rceil}}^{\text{bin}(k)} | \underbrace{0\dots0}_{\lceil\log(k)\rceil} | \overbrace{\underbrace{1111\dots1111}_{k}}^{\text{bin}(2^{k+1}-1)} | \underbrace{0\dots0}_{\lceil\log(k)\rceil} | \overbrace{\underbrace{1011\dots}_{\lceil\log(T)\rceil}}^{\text{bin}(T)}$$

Figure 3: Construction of Z_i' and T'

For $m > 1$ we can use the same construction as in the reduction for the optimization problem: Suppose there is an algorithm that finds feasible solutions to ILPs with m constraints in time $n^{f(m)} \cdot (\Delta + \|b\|_\infty)^{m-\delta}$. Choose γ such that there is no algorithm for k-SUM with running time $O(T^{1-\delta/m} \cdot n^{\gamma k})$ (under SETH). We set $k = \lceil f(m)/\gamma \rceil$. By splitting the one constraint of $(*)$ into m constraints we can reduce the upper bound on elements from $O(T)$ to $O(T^{1/m})$. This means the assumed running time for solving ILPs can be used to solve k-SUM in time

$$n^{f(m)} \cdot O(T^{1/m})^{m-\delta} \leq n^{\gamma k} O(1)^{m-\delta} T^{1-\delta/m} = O(n^{\gamma k} T^{1-\delta/m}). \qquad \square$$

2.5 APPLICATIONS

We describe the implications of our results on a couple of well-known problems, which can be formulated using ILPs with few constraints and small entries. In particular, we give an example, where the reduction of the running time by a factor n improves on the state-of-the-art and one where the logarithmic dependence on $\|b\|_\infty$ proves useful.

Unbounded Knapsack and Unbounded Subset-Sum

UNBOUNDED KNAPSACK with equality constraint is simply an ILP with $m = 1$ and positive entries and objective function:

$$\max\{\sum_{i=1}^{n} p_i \cdot x_i : \sum_{i=1}^{n} w_i \cdot x_i = C, x \in \mathbb{Z}^n_{\geq 0}\}$$

where $p_i \geq 0$ are called the profits and $w_i \geq 0$ the weights of the items $1, \dots, n$. More common is to let C be only an upper bound on $\sum_{i=1}^{n} w_i \cdot x_i$, but that variant easily reduces to the problem above by adding a slack variable. UNBOUNDED SUBSET-SUM is the same problem without an objective function, i.e., the problem of finding a multiset of items whose weights sum up to exactly C. We assume that no two items have the same weight. Otherwise in time $O(n + \Delta)$ we can remove all duplicates by keeping only the

most valuable ones. The fractional solutions to both problems are of a very simple structure: For Unbounded Knapsack choose only the item i of maximal efficiency, that is p_i/w_i, and select it C/w_i times. For Unbounded Subset-Sum choose an arbitrary item. This gives algorithms with running time $O(\Delta^2)$ and $O(\Delta \log^2(\Delta))$ for Unbounded Knapsack and Unbounded Subset-Sum, respectively, where Δ is the maximum weight among all items (using the results from Section 2.3). The previously best pseudo-polynomial algorithms for Unbounded Knapsack, have running times $O(nC)$ (standard dynamic programming; see e.g. [49]), $O(n\Delta^2)$ [30], or very recently $O(\Delta^2 \log(C))$ [10]. We note that the algorithm from the last publication, which was discovered independently and concurrently to our results, also uses (min, +)-convolution. It could probably be improved to the same running time as our general algorithm using the proximity ideas.

For Unbounded Subset-Sum the state-of-the-art is a $O(C \log(C))$ time algorithm [17]. Hence, our algorithm is preferable when $\Delta \ll C$.

Scheduling on Identical Machines

The problem Scheduling on Identical Machines asks for the distribution of N jobs onto $M \leq N$ machines. Each job j has a processing time p_j and the objective is to minimize the makespan, i.e., the maximum sum of processing times on a single machine. Since an exact solution cannot be computed unless P = NP, we are satisfied with a $(1+\epsilon)$-approximation, where $\epsilon > 0$ is part of the input. We will outline how this problem can be solved using our algorithm. More details on many of the techniques involved can be found in [37].

We consider here the variant, in which a makespan τ is given and we have to find a schedule with makespan at most $(1+\epsilon)\tau$ or prove that there exists no schedule with makespan at most τ. This suffices by using a standard dual approximation framework. It is easy to see that one can discard all jobs of size at most $\epsilon \cdot \tau$ and add them greedily after a solution for the other jobs is found. The big jobs can each be rounded to the next value of the form $\epsilon \cdot \tau \cdot (1+\epsilon)^i$ for some i. This reduces the number of different processing times to $O(1/\epsilon \log(1/\epsilon))$ many and increases the makespan by at most a factor of $1+\epsilon$. We are now ready to write this problem as an ILP. A configuration is a way to use a machine. It describes how many jobs of each size are assigned to this machine. Since we aim for a makespan of $(1+\epsilon) \cdot \tau$, the sum of these sizes must not exceed this value. The configuration ILP has a variable for every valid configuration and it describes how many machines use this configuration. Let $\mathcal{C}$ be the set of valid configurations and C_k the multiplicity

of size k in a configuration $C \in \mathcal{C}$. The following ILP solves the rounded instance. We note that there is no objective function in it.

$$\begin{aligned} \sum_{C \in \mathcal{C}} x_C &= M \\ \sum_{C \in \mathcal{C}} C_k \cdot x_C &= N_k && \forall k \in \mathcal{K} \\ x_C &\in \mathbb{Z}_{\geq 0} && \forall C \in \mathcal{C} \end{aligned}$$

Here $\mathcal{K}$ are the rounded sizes and N_k the number of jobs with rounded size $k \in \mathcal{K}$. The first constraint enforces that the correct number of machines is used, the next $|\mathcal{K}|$ many enforce that for each size the correct number of jobs is scheduled.

It is notable that this ILP has only few constraints (a constant for a fixed choice of ϵ) and also the ℓ_1 norm of each column is small. More precisely, it is at most $1/\epsilon$, since every size is at least $\epsilon \cdot \tau$ and therefore no more than $1/\epsilon$ jobs fit in one configuration. The ILP can be solved with our algorithm. By the Beck-Fiala Theorem (Theorem 5) $H = 1/\epsilon$ is an upper bound on the hereditary discrepancy, $\Delta \leq 1/\epsilon$, $m = O(1/\epsilon \log(1/\epsilon))$, $\|b\|_\infty \leq N$, and $n \leq (1/\epsilon)^{O(1/\epsilon \log(1/\epsilon))}$. Including the rounding in time $O(N + 1/\epsilon \log(1/\epsilon))$ the running time for the ILP is

$$\begin{aligned} &O(H)^m \log(\Delta) \log(\Delta + \|b\|_\infty) + O(nm) + O(N + \frac{1}{\epsilon} \log(\frac{1}{\epsilon})) \\ &\leq 2^{O(1/\epsilon \log^2(1/\epsilon))} \log(N) + O(N + \frac{1}{\epsilon} \log(\frac{1}{\epsilon})) \\ &\leq 2^{O(1/\epsilon \log^2(1/\epsilon))} + O(N). \end{aligned}$$

The trick in the bound above is to distinguish between $2^{O(1/\epsilon \log^2(1/\epsilon))} \leq \log(N)$ and $2^{O(1/\epsilon \log^2(1/\epsilon))} > \log(N)$. The same running time (except for a higher constant in the exponent) could be obtained with [30]. However, in order to avoid a multiplicative factor of N, one would have to solve the LP relaxation first and then use proximity. Our approach gives an easier, purely combinatorial algorithm. The crucial feature of our algorithm is the lower dependence on $\|b\|_\infty$.

In this chapter we extend the setting from the previous chapter to Mixed Integer Linear Programs. This is unpublished joint work with José Verschae from a visit in Kiel.

3

PSEUDO-POLYNOMIAL MILP

It is well known that linear programs can be solved efficiently (in polynomial time). Integer linear programs are NP-hard, but the algorithms by Lenstra [47] and Kannan [48] can solve them in polynomial time, if the number of variables is constant. Moreover, the intermediate is also solvable: Already in [47], Lenstra pointed out that mixed integer linear programs, which contain both integer and fractional variables, can also be solved in polynomial time if the number of integer variables is a constant.

Theorem 14 ([47, 48]). *An MILP with m integer variables can be solved in time $m^{O(m)} \cdot |I|^{O(1)}$, where $|I|$ is the encoding size of the MILP.*

The previous chapter shows that integer linear programs with a constant number of constraints can be solved in pseudo-polynomial time (regarding the size of the entries in the matrix). This gives rise to a natural question: What is the complexity of mixed integer linear programs where only a constant number of constraints have non-zero entries in the integer variables. In particular, is there a pseudo-polynomial algorithm in the size of the entries of the integer variables? Formally, let $A \in \mathbb{Z}^{r\times m}$, $B \in \mathbb{Z}^{r\times n}$, and $C \in \mathbb{Z}^{s\times n}$. We are looking for a solution of the system

$$\begin{aligned} &\max\ c^T x + d^T y \\ &\begin{pmatrix} A & B \\ 0 & C \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = b \\ &\qquad x \in \mathbb{Z}^m_{\geq 0} \\ &\qquad y \in \mathbb{R}^n_{\geq 0}. \end{aligned} \tag{8}$$

Furthermore, let Δ be an upper bound on the absolute value of the entries of A (not necessarily on B or C). In this chapter, we will prove the following.

Theorem 15. *There is an algorithm that solves the system above in time*

$$(r\Delta)^{O(r^2)} \cdot |I|^{O(1)},$$

where $|I|$ is the encoding length of the MILP (8).

We are not aware of directly comparable results from literature. We will start by providing a simple algorithm using standard techniques and then

give our improved algorithm. Throughout the chapter assume w.l.o.g. that $m \leq (2\Delta+1)^r$. This is the maximum number of distinct columns of A and when duplicate columns exist, the one with the lower objective value can be ignored.

A SIMPLE ALGORITHM. Let x, y be an optimal solution to the MILP (8). Let $b' = Ax$. Then clearly x is an optimal solution for the ILP

$$\begin{aligned} \max\ & c^T z \\ & Az = b' \\ & z \in \mathbb{Z}_{\geq 0}. \end{aligned} \tag{9}$$

We will use a theorem that gives a bound on the smallest support (i.e., non-zero components) of an optimal solution of this ILP.

Theorem 16 ([2]). *Consider the ILP (9). If it is bounded and feasible, then there exists an optimal solution x' with support of cardinality at most*

$$2r\log(2\sqrt{r}\Delta) = O(r\log(r\Delta)).$$

Let x' as in the theorem. Then x', y is still an optimal solution for (8). Furthermore, we can guess the support of x' in time

$$m^{O(r\log(r\Delta))} \leq \Delta^{O(r^2\log(r\Delta))} = (r\Delta)^{O(r^2\log(\Delta))}.$$

If we remove all integer variables outside the support of x', this does not affect the optimum of the MILP. It remains to solve an MILP with only $O(r\log(r\Delta))$ integer variables. This can be solved using Kannan's algorithm in time

$$(r\log(r\Delta))^{O(r\log(r\Delta))} \cdot |I|^{O(1)}.$$

Putting everything together this yields a total running time of

$$\begin{aligned} (r\Delta)^{O(r^2\log(\Delta))} \cdot (r\log(r\Delta))^{O(r\log(r\Delta))} \cdot |I|^{O(1)} \\ \leq (r\Delta)^{O(r^2\log(\Delta))} \cdot |I|^{O(1)}. \end{aligned}$$

For a constant r, this is quasi-polynomial in Δ. In fact, both guessing the support and solving the reduced MILP require super-polynomial time in Δ.

A MORE SOPHISTICATED GUESSING APPROACH. Consider again the ILP (9) and let x^* be an optimal vertex solution for its continuous relaxation. By Theorem 9 from the last chapter, we know that there exists an integer optimal solution x' with

$$\|x' - x^*\|_1 \leq k,$$

where $k = O(r\Delta)^r$. Since x^* is a vertex solution, the size of its support is bounded by r. Let $x'' \in \mathbb{Z}^m_{\geq 0}$ with $x''_i = \max\{0, \lceil x^*_i - k \rceil\}$. By triangle inequality we get $\|x' - x''\|_1 \leq (r+1)k$ and also $x' - x''$ is non-negative.

Let $b'' = A(x'-x'')$. Note that b'' is integer-valued and $\|b''\|_\infty \leq (r+1)k\Delta$. Furthermore, $x'-x''$ is a feasible solution of the ILP

$$\begin{aligned} \max\ & c^T z \\ & Az = b'' \\ & z \in \mathbb{Z}_{\geq 0}, \end{aligned} \tag{10}$$

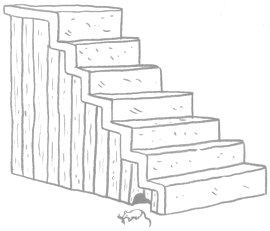
and x'', y is a feasible solution of the MILP

$$\begin{aligned} \max\ & c^T z + d^T w \\ & \begin{pmatrix} A & B \\ 0 & C \end{pmatrix} \begin{pmatrix} z \\ w \end{pmatrix} = b - b'' \\ & z \in \mathbb{Z}^m_{\geq 0} \\ & w \in \mathbb{R}^n_{\geq 0}. \end{aligned} \tag{11}$$

It is easy to see that when we add solutions of (10) and (11) we get a solution for the original MILP. If either $x'-x''$ or x'', y were not optimal solutions, then x', y would not be optimal for the original MILP, which it is. Our approach now is to guess b'' and solve (10) and (11) individually. Indeed, we have reduced solving an MILP to solving an ILP and a very similar MILP. Did we gain anything? Surprisingly, (11) is easier to solve. Recall, that x'', y is an optimal solution and the support of x'' is at most r. This reduces the running time of the previous approach to a pseudo-polynomial one. In time $((r+1)k\Delta)^r = (r\Delta)^{O(r^2)}$ we guess the value of b''; in time $O(\Delta)^{r^2}$ we guess the correct support of x''; and in time $r^{O(r)} \cdot |I|^{O(1)}$ we solve the reduced MILP (11). This gives an overall running time of $(r\Delta)^{O(r^2)} \cdot |I|^{O(1)}$.

CONCLUDING REMARKS. Interestingly, the best algorithm for ILPs with a constant number of constraints, where in addition upper bounds on the variables (not counted as constraints) are given, has a very similar running time, i.e., also a quadratic dependence in the exponent [30]. For both this problem and the MILP problem it is unclear whether this quadratic dependency is necessary or there exists an algorithm with a linear exponent. Another question would be to unify both extensions, i.e., to get a pseudo-polynomial algorithm for MILPs with upper bounds on the variables.

Our algorithm uses the heavy machinery of Kannan's algorithm, which in turn uses the Ellipsoid method. Is this necessary or is there a simpler way? We argue that this complexity is justified, since pseudo-polynomiality for this MILP implies polynomiality for both linear programming and integer programming with a constant number of variables. Clearly, setting $m = 0$ makes the problem a classical linear program. Hence, the algorithm should be at least as complicated as an LP solver. Furthermore, by setting A to the identity matrix, B to its negation, and the first r entries of b to 0 we can reduce an arbitrary integer program with r variables (and matrix C) to this MILP with $\Delta = 1$. Note that this construction enforces integrality on the (fractional) variables y.



The previous chapters were restricted to ILPs with a constant number of constraints on integer variables. Here we give an algorithm for many constraints, when the matrix has a certain block structure.

4

n-FOLD INTEGER PROGRAMMING

We consider n-fold ILPs, a class of ILPs that have many constraints and variables and can still be solved in pseudo-polynomial time. The characteristic feature of an n-fold ILP is the block structure of its matrix $\mathcal{A}$. Non-zero entries appear only in the first r rows of $\mathcal{A}$ and in blocks of size $s \times t$ along the diagonal underneath. More precisely, the constraint matrix in an n-fold ILP has the form

$$\mathcal{A} = \begin{pmatrix} A_1 & A_2 & \dots & A_n \\ B_1 & 0 & \dots & 0 \\ 0 & B_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & B_n \end{pmatrix},$$

where $A_1, \dots, A_n$ are $r \times t$ matrices and $B_1, \dots, B_n$ are $s \times t$ matrices. In n-fold ILPs we also allow upper and lower bounds on the variables. Throughout the chapter we subdivide a solution x into bricks of length t and denote by $x^{(i)}$ the i-th one. The corresponding columns in $\mathcal{A}$ will be called blocks.

Lately, n-fold ILPs received great attention [4, 29, 36, 51, 54] and were studied intensively due to two reasons. Firstly, many optimization problems are expressible as n-fold ILPs [26, 33, 36, 51]. Secondly, n-fold ILPs indeed can be solved much more efficiently than arbitrary ILPs [29, 33, 54]. The previously best algorithm has a running time of $(rs\Delta)^{O(r^2s+rs^2)} L \cdot (nt)^2 \log^2(n \cdot t) + \mathrm{LP}$ and is due to Eisenbrand et al. [29]. Here LP is the running time required for solving the corresponding LP relaxation. This augmentation algorithm is the last one in a line of research, where local improvement/augmenting steps are used to converge to an optimal solution. Clever insights about the structure of the improving directions allow them to be computed fast. Nevertheless, the dependence on n in the algorithm above is still high. Indeed, in practice a quadratic running time is simply not suitable for large data sets [5, 27, 50]. For example when analyzing big data, large real world graphs as in telecommunication networks or DNA strings in biology, the duration of the computation would go far beyond the scope of an acceptable running time [5, 27, 50]. For this reason even problems which have an algorithm of quadratic running time are still studied from the viewpoint of approximation algorithms with

the objective to obtain results in sub-quadratic time, even at the cost of a worse quality [5, 27, 50]. Hence, it is an intriguing question, whether the quadratic dependency on the number of variables, i.e., nt, can be eliminated. In this chapter, we answer this question affirmatively. The technical novelty comes from a surprising area: We use a combinatorial structure called splitter, which has been used to derandomize Color Coding algorithms. It allows us to build a powerful data structure that is maintained during the local search and from which we can derive an improving direction in logarithmic time. Handling unbounded variables in an n-fold ILP is a non-trivial issue in the previous algorithms from literature. They had to solve the corresponding LP relaxation and use proximity results. Unfortunately, it is not known whether linear programming can be solved in near-linear time in the number of variables. Hence, it is an obstacle for obtaining a near-linear running time. We manage to circumvent the necessity of solving the LP by introducing artificial bounds as a function of the finite upper bounds and the right-hand side of the n-fold ILP.

SUMMARY OF RESULTS

- We present an algorithm, which solves n-fold ILPs in time

  $$(rs\Delta)^{O(r^2s+s^2)}L \cdot nt \log^5(nt) + \mathrm{LP},$$

  where LP is the time to solve the LP relaxation of the n-fold ILP. This is the first algorithm with a near-linear dependence on the number of variables. The crucial step is to speed up the computation of the improving directions.

- We circumvent the need for solving the LP relaxation. This leads to a purely combinatorial algorithm with running time

  $$(rs\Delta)^{O(r^2s+s^2)}L^2 \cdot nt \log^7(nt).$$

- In the running times above the dependence on the parameters, i.e., $(rs\Delta)^{O(r^2s+s^2)}$, improves on the function $(rs\Delta)^{O(r^2s+rs^2)}$ in the previous best algorithms.

OUTLINE OF NEW TECHNIQUES. We will briefly elaborate our main technical novelty. Let x be some feasible, non-optimal solution for the n-fold ILP. It is clear that when y^* is an optimal solution for

$$\max\{c^T y \mid \mathcal{A}y = 0, \ell - x \leq y \leq u - x, y \in \mathbb{Z}^{nt}\},$$

then $x + y^*$ is optimal for the initial n-fold ILP. In other words, y^* is a particularly good improving step. A sensible approximation of y^* is to consider directions y of small size and multiplying them by some step length, i.e., find some $\lambda \cdot y$ with $\|y\|_1 \leq k$ for a value k depending only on Δ, r, and s. This implies that at most k of the n bricks are used for y. If we randomly color the blocks with k^2 colors, then with high probability at most one block of every

color is used. This reduces the problem to choosing a solution of a single brick for every color and to aggregate them. We add data structures for every color to implement this efficiently. There is of course a chance that the colors do not split y perfectly. We handle this by using a deterministic structure of multiple colorings (instead of one) such that it is guaranteed that at least one of them has the desired property.

RELATED WORK. The first XP-time algorithm for solving n-fold integer programs is due to De Loera et al. [26] with a running time of $n^{g(\mathcal{A})}L$. Here $g(\mathcal{A})$ denotes a so-called Graver complexity of the constraint matrix $\mathcal{A}$ and L is the encoding length of the largest number in the input. This algorithm already uses the idea of iterative converging to the optimal solution by finding improving directions. Nevertheless, the Graver complexity appears to be huge even for small n-fold integer linear programs and thus this algorithm was of no practical use [33]. The exponent of this algorithm was then greatly improved by Hemmecke et al. [33] to a constant factor yielding the first cubic time algorithm for solving n-fold ILPs. More precisely, the running time of their algorithm is $\Delta^{O(t(rs+st))}L \cdot (nt)^3$, i.e., FPT-time parameterized over $\Delta, r, s,$ and t. Lately, two more breakthroughs were obtained. One of the results is due to Koutecký et al. [54], who gave a strongly polynomial algorithm with running time $\Delta^{O(r^2s+rs^2)}(nt)^6 \cdot \log(nt) + \mathrm{LP}$. Here LP is the running time for solving the corresponding LP relaxation, which is possible in strongly polynomial time, since the entries of the matrix are bounded. Simultaneously, Eisenbrand et al. reduced in [29] the running time from a cubic factor to a quadratic one by introducing new proximity and sensitivity results. This leads to an algorithm with running time $(\Delta rs)^{O(r^2s+rs^2)}L \cdot (nt)^2 \log^2(nt) + \mathrm{LP}$. Note that both results require only polynomial dependency on t.

As for applications, n-fold ILPs are broadly used to model various problems such as string, flow or scheduling problems. We refer to the works [26, 33, 34, 36, 52, 59] and the references therein for an overview.

STRUCTURE OF THE CHAPTER. In Section 4.1 we introduce the necessary preliminaries. Section 4.2 gives the algorithm for efficiently computing the augmenting steps. This is then integrated into an algorithm for n-fold ILPs in Section 4.3. At first we require finite variable bounds and then discuss how to eliminate this requirement using the solution of the LP relaxation. Finally, in Section 4.4 we discuss how to handle infinite variable bounds without the LP relaxation and give new structural results.

4.1 PRELIMINARIES

In the following we introduce n-fold ILPs formally and state the main results regarding them. Further we familiarize splitters, a technique known from Color Coding.

Definition 3. *Let $n, r, s, t \in \mathbb{N}$. Furthermore let $A_1, \ldots, A_n$ be $r \times t$ integer matrices and $B_1, \ldots, B_n$ be $s \times t$ integer matrices. Then the n-fold matrix $\mathcal{A}$ is of following form:*

$$\mathcal{A} = \begin{pmatrix} A_1 & A_2 & \ldots & A_n \\ B_1 & 0 & \ldots & 0 \\ 0 & B_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & B_n \end{pmatrix}.$$

The matrix $\mathcal{A}$ is of dimension $(r + n \cdot s) \times n \cdot t$. We will divide $\mathcal{A}$ into blocks of size $(r + n \cdot s) \times t$. Similarly, the variables of a solution x are partitioned into bricks of length t. This means each brick $x^{(i)}$ corresponds to the columns of one submatrix A_i and therefore also B_i. Given $c, \ell, u \in \mathbb{Z}^{n \cdot t}$ and $b \in \mathbb{Z}^{r+n \cdot s}$, the corresponding n-fold Integer Linear Programming problem is defined by:

$$\max\{c^T x \mid \mathcal{A}x = b, \ell \leq x \leq u, x \in \mathbb{Z}^{n \cdot t}\}.$$

The main idea for the state-of-the-art algorithms relies on some insight about the Graver basis of n-fold ILPs, which are special elements of the kern of $\mathcal{A}$. More formally, we introduce the following definitions:

Definition 4. *The kern of a matrix A is defined as the set of integral vectors $x \in \mathbb{Z}^{\cdot}$ with $Ax = 0$. We write $\ker(A)$ for them.*

Definition 5. *A Graver basis element g is a minimal element of $\ker(A)$. An element is minimal, if it is not the sum of two sign-compatible elements $u, v \in \ker(A)$.*

Here, sign-compatible means that $u_i \cdot v_i \geq 0$ for every i.

Theorem 17 ([24])**.** *Let $A \in \mathbb{Z}^{n \times m}$ and let $x \in \ker(A)$. Then there exist $2n - 1$ Graver basis elements $g_1, \ldots, g_{2n-1}$, which are sign-compatible with x such that*

$$x = \sum_{i=1}^{2n-1} \lambda_i g_i$$

for some $\lambda_1, \ldots, \lambda_{2n-1} \in \mathbb{Z}_{\geq 0}$.

Many results for n-fold ILPs rely on the fact that the ℓ_1-norm of Graver basis elements for n-fold matrices are small. The best bound known for the ℓ_1-norm is due to Eisenbrand et al. [29].

Theorem 18 ([29])**.** *The ℓ_1-norm of the Graver basis elements of an n-fold matrix $\mathcal{A}$ is bounded by $O(rs\Delta)^{rs}$.*

Next, we will introduce a technique called splitters (see e.g. [57]), which has its origins in the FPT community and was used to derandomize the Color Coding technique [3]. So far it has not been used with n-fold ILPs. We refer the reader to the outline of techniques in the introduction for the idea on how we apply the splitters.

Definition 6. *An (n,k,ℓ) splitter is a family of hash functions F from the numbers $\{1,\dots,n\}$ to $\{1,\dots,\ell\}$ such that for every $S \subseteq \{1,\dots,n\}$ with $|S| = k$, there exists a function $f \in F$ that splits S evenly, that is, for every $j, j' \leq \ell$ we have $|f^{-1}(j) \cap S|$ and $|f^{-1}(j') \cap S|$ differ by at most 1.*

If $\ell \geq k$, the above means that there is some hash function that has no collisions when restricted to S. Interestingly, there exist splitters of very small size.

Theorem 19 ([3, 58])**.** *There is an (n,k,k^2) splitter of size $k^6 \log(k) \log(n)$ which is computable in time $k^6 \log(k) \cdot n \log(n)$.*

We note that an alternative approach to the result above is to use FKS hashing. Although it has an extra factor of $\log(n)$, it is particularly easy to implement.

Theorem 20 (Corollary 2 and Lemma 2, [32])**.** *Define for every pair of primes $p < q < k^2 \log(n)$ the hash function $x \mapsto 1 + (p \cdot (x \bmod q) \bmod k^2)$. This is an (n,k,k^2) splitter of size $O(k^4 \log^2(n))$.*

We remark that a hash function from $\{1,\dots,n\}$ to $\{1,\dots,\ell\}$ naturally corresponds to a partition of the set $\{1,\dots,n\}$ into exactly ℓ subsets.

4.2 EFFICIENT COMPUTATION OF IMPROVING DIRECTIONS

The backbone of our algorithm is the efficient computation of augmenting steps. The important aspect is the fact that we can update the augmenting steps very efficiently if the input changes only slightly. In other words, whenever we change the current solution by applying an augmenting step, we do not have to recompute the next augmenting step from scratch. The augmenting steps depend on a partition of the bricks. In the following we define the notion of a best step based on a fixed partition. Later, we will independently find steps for a number of partitions and take the best among them.

Definition 7. *Let P be a partition of the n bricks into the k^2 disjoint sets $P_1, P_2, \dots, P_{k^2}$. Let $\overline{u} \in \mathbb{Z}^{nt}_{\geq 0}$ and $\overline{\ell} \in \mathbb{Z}^{nt}_{\leq 0}$ be some upper and lower bounds on the variables (not necessarily the same as in the n-fold ILP). A (P,k)-best step is an optimal solution of the system below. We slightly abuse notation by using P_j or bricks $S_j \in P_j$ for the indices of variables contained in them.*

$$
\begin{aligned}
\max\ & c^T x && \\
& \mathcal{A}x = 0 && \\
& \sum_{i \in S_j} |x_i| \leq k && \forall j \in \{1,\dots,k^2\} \\
& x_i = 0 && \forall j \in \{1,\dots,k^2\}, i \in P_j \setminus \{S_j\} \\
& \overline{\ell} \leq x \leq \overline{u} && \\
& x \in \mathbb{Z}^{nt} && \\
& S_j \in P_j && \forall j \in \{1,\dots,k^2\}.
\end{aligned}
$$

This means a (P,k)-best step is an element of $\ker(\mathcal{A})$, which uses only one brick of every $P_j \in P$. Within that brick the norm of the solution must be at most k. Later, we will choose k as the upper bound for the ℓ_1-norm of a Graver basis element, i.e., $k = O(rs\Delta)^{rs}$.

Lemma 21. *Consider the problem of finding a (P,k)-best step in an n-fold matrix where the lower and upper bounds $\overline{u},\overline{\ell}$ can change. This problem can be solved initially in time $k^{O(r)} \cdot \Delta^{O(r^2+s^2)} \cdot nt$ and then in $k^{O(r)} \cdot \Delta^{O(r^2+s^2)} \cdot \log(nt)$ update time whenever the bounds of a single variable change.*

Proof. Let P be a partition of the bricks from matrix $\mathcal{A}$ into k^2 disjoint sets $P_1, P_2, \ldots, P_{k^2}$. Solving the (P,k)-best step problem requires that from each set $P_j \in P$ we choose at most one brick and set this brick's variables. All variables in other bricks of P_j must be 0.

Let x be a (P,k)-best step and let $x^{(j)}$ have the values of x in variables of P_j and 0 in all other variables. Then by definition, $\|x^{(j)}\|_1 \leq k$. This implies that the right-hand side regarding $x^{(j)}$, that is to say, $\mathcal{A}x^{(j)}$, is also small. Since the absolute value of an entry in $\mathcal{A}$ is at most Δ, we have that $\|\mathcal{A}x^{(j)}\|_\infty \leq k\Delta$. Let a_i be the i-th row of $\mathcal{A}$. If $i > r$, then $a_i x^{(j)} = 0$. This is because $\mathcal{A}x = 0$ and a_i has all its support either completely inside P_j or completely outside P_j. Meaning, the value of $\mathcal{A}x^{(j)}$ is one of the $(2k\Delta+1)^r$ many values we get by enumerating all possibilities for the first r rows. Furthermore, since P has only k^2 sets, the partial sum $\mathcal{A}(x^{(1)} + \cdots + x^{(j)})$ is always one of $(2k^3\Delta+1)^r = (k\Delta)^{O(r)}$ many candidates.

Hence to find a (P,k)-best step we can restrict our search to solutions whose partial sums stay in this range. To do so, we set up a graph containing k^2+2 layers $L_0, L_1, \ldots L_{k^2}, L_{k^2+1}$. An example is given in figure 4. The first layer L_0 will consist of just one node marking the starting point with partial sum zero. Similarly, the last layer L_{k^2+1} will just contain the target point also having partial sum zero, since a (P,k)-best step is an element of $\ker(\mathcal{A})$. Each layer L_j with $1 \leq j \leq k^2$ will contain $(2k^3\Delta+1)^r$ many nodes, each representing one possible value of $\mathcal{A}(x^{(1)} + \cdots x^{(j)})$. Two points v, w from adjacent layers L_{j-1}, L_j will be connected if the difference of the corresponding partial sums, namely $w - v$, can be obtained by a solution y of variables from only one brick of P_j (with $\|y\|_1 \leq k$). The weight of the edge will be the largest gain for the objective function $c^T y$ over all possible bricks. Hence, it could be necessary to compute and compare up to n values for each P_j and each difference in the partial sums to insert one edge into the graph. Finally, we just have to find the longest path in this graph as it corresponds to a (P,k)-best step. The out-degree of each node is bounded by $(2k^3\Delta+1)^r$ since at most this many nodes are reachable in the next layer. Therefore the overall number of edges is bounded by

$$(k^2+2) \cdot (2k^3\Delta+1)^r \cdot (2k^3\Delta+1)^r = (k\Delta)^{O(r)}.$$

Using the Bellman-Ford algorithm we can solve the Longest Path problem for a graph with N vertices and M edges in time $O(N \cdot M)$ as the graph is directed and acyclic. This gives a running time of $(k\Delta)^{O(r)} \cdot (k\Delta)^{O(r)} = (k\Delta)^{O(r)}$ for

solving the problem. Constructing the graph, however, requires solving a number of IPs of the form

$$\begin{aligned} &\max c'^T x \\ &\begin{pmatrix} A_j \\ B_j \end{pmatrix} x = \begin{pmatrix} b' \\ 0 \end{pmatrix} \\ &\|x\|_1 \leq k \\ &\ell' \leq x \leq u' \\ &x \in \mathbb{Z}^t, \end{aligned}$$

where $b' \in \mathbb{Z}^r$ is the corresponding right-hand side of the top rows and ℓ', u', c' are the upper and lower bounds, and the objective of the block. This is an IP with $r+s+1$ constraints, t variables, lower and upper bounds, and entries of the matrix bounded by Δ in absolute value. Using the algorithm by Eisenbrand and Weismantel [30], solving one of them requires time

$$t \cdot O(r+s+1)^{r+s+4} \cdot O(\Delta)^{(r+s+1)(r+s+4)} \cdot \log^2((r+s+1)\Delta) + \mathrm{LP},$$

where LP is the time for solving the LP relaxation. This simplifies to $t \cdot \Delta^{O(r^2+s^2)} + \mathrm{LP}$ when we do not care about the constant in the exponent. Note that strictly speaking the constraint $\|x\|_1 \leq k$ is not linear, but by standard construction the ILP can easily be transformed into an equivalent one replacing every variable x_i with a composition $x_i^+ - x_i^-$ of two positive variables. Then the ℓ_1-norm is simply the sum over all variables. This does not affect the asymptotic running time.

Furthermore, a little thought allows us to reduce the dependency on t to a logarithmic one: Since the number of constraints in the ILP above is very small, there are only $\Delta^{O(r+s)}$ many different columns. Because of the cardinality constraint $\|x\|_1 \leq k$, we only have to consider $2k$ many variables of each type of column, namely: The k many with $u_i' > 0$ and maximal c_i' and the k many with $\ell_i' < 0$ and minimal c_i'.

If some solution uses a variable not in this set, then by pigeonhole principle there is a variable with the same column values and a superior objective value and which can be increased/decreased. We can reduce the variable outside this set and increase the corresponding variable inside this set until all variables outside the set are 0. We can use an appropriate data structure (e.g. AVL trees) to maintain a set of all variables with $u_i' > 0$ ($\ell_i' < 0$) such that we can find the k best among them in time $O(k \log(t))$. Whenever the bounds of some variable change, we might have to add or remove entries, which also takes only logarithmic time. After initialization in time $O(nt)$ (in total for all bricks) solving such an IP can therefore be implemented in time

$$\begin{aligned} &k\log(t) + 2k\Delta^{O(r+s)}\Delta^{O(r^2+s^2)} + \mathrm{LP} \\ &\qquad \leq k\log(t)\Delta^{O(r^2+s^2)} + \mathrm{LP} \leq k\log(t)\Delta^{O(r^2+s^2)}. \end{aligned}$$

The last inequality holds, because using Tardos' algorithm [69] LPs can be solved in time polynomial in the encoding size of the matrix, which can be

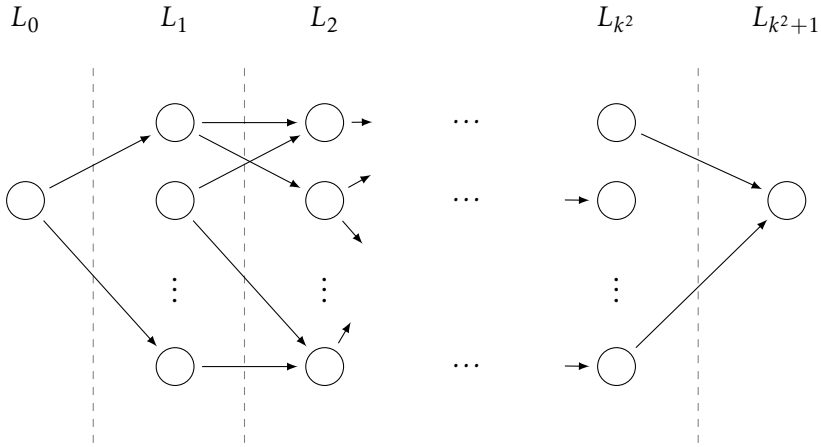


Figure 4: This figure shows an example for a layered graph obtained while solving the (P,k)-best step problem. There are k^2+2 layers, visually separated by gray dashed lines. This includes one source layer L_0, one target layer L_{k^2+1} both with just a single node representing the zero sum. Further there are k^2 layers with $(2k^3\Delta+1)^r$ nodes each, where in one layer the nodes stand for all reachable partial sums. Two points v,w from adjacent layers L_{j-1}, L_j will be connected if the difference of the corresponding partial sums, namely $w-v$, can be obtained by a solution y of variables from only one brick of P_j (with $\|y\|_1 \leq k$). The weight of the edge will be the largest gain for the objective function $c^T y$ over all possible bricks. For the sake of clarity both the values of the nodes and the edges are not illustrated.

bounded by $2k\Delta^{O(r+s)} \cdot (r+s) \cdot \log(\Delta)$. This is dominated by the other term. The number of IPs to solve is at most n times the number of edges, since we have to compare the values of up to n bricks. This gives a running time of

$$O(nt) + n \cdot (k\Delta)^{O(r)} \cdot \log(t) \cdot \Delta^{O(r^2+s^2)} \leq nt \cdot k^{O(r)} \cdot \Delta^{O(r^2+s^2)}$$

for constructing the graph. To obtain the update time from the premise of the theorem, it is perfectly fine to solve the Longest Path problem again, but we cannot construct the graph from scratch. However, in order to construct the graph we still have to find the best value over all bricks for each edge. Fortunately, if only a few bricks are updated (in their lower and upper bounds) it is not necessary to recompute all values. Each edge corresponds to a particular $P_j \in P$ and a fixed right-hand side (a possible value of $\mathcal{A}x^{(j)}$). We require an appropriate data structure $\mathcal{D}_e$ for every edge e, which supports fast computation of the operations FindMax, Insert, and Delete. Again, an AVL tree computes each of these operations in time $O(\log(N))$, where N is the number of elements. In $\mathcal{D}_e$ we store pairs (v,i) where i is a brick in P_j and v is the maximum gain of brick i for the right-hand side of e. The pairs are stored in lexicographical order. Since there are at most n bricks in P_j, the data structure will have at most n elements. Initially, we can build $\mathcal{D}_e$ in time $nt \cdot \Delta^{O(r^2+s^2)}$ (this is replicated for each edge). Now consider a change to the instance. Recall that we are looking at changes that affect only a single brick, namely the upper and lower bounds within that brick change.

We are going to update the data structure $\mathcal{D}_e$ (for each edge) to reflect the changes and we are going to recompute the edge value of each edge e using $\mathcal{D}_e$. Then we simply solve the Longest Path problem again. Let $P_j \in P$ be the set that contains the brick i that has changed in some variable. We only have to consider edges from L_{j-1} to L_j, since none of the other edges are affected by the change. For a relevant edge e we compute the previous value v and current value v' that the brick i would produce (before and after the bounds have changed). In $\mathcal{D}_e$ we have to remove (v,i) and insert (v',i). Both operations need only $O(\log(n))$ time. Then the running time to update $\mathcal{D}_e$ for one edge is

$$k\log(t)\cdot\Delta^{O(r^2+s^2)} + O(\log(n)) \leq k\log(nt)\cdot\Delta^{O(r^2+s^2)}.$$

In order to update the edge value of e using $\mathcal{D}_e$, we simply have to find the maximum element in $\mathcal{D}_e$, which again takes time $O(\log(n))$. To summarize, the total time to update the (P,k)-best step after a change to a single brick consists of (1) updating each $\mathcal{D}_e$, (2) finding the maximum in each $\mathcal{D}_e$, and (3) solving the Longest Path problem. We conclude that the update time is

$$\begin{aligned} k\log(nt)\cdot\Delta^{O(r^2+s^2)}\cdot(k\Delta)^{O(r)} + \log(n)\cdot(k\Delta)^{O(r)} + (k\Delta)^{O(r)} \\ \leq k^{O(r)}\Delta^{O(r^2+s^2)}\cdot\log(nt). \quad \square \end{aligned}$$

4.3 THE AUGMENTING STEP ALGORITHM

In this section we will assume that all lower and upper bounds are finite and give a complete algorithm for this case. Later, we will explain how to cope with infinite bounds. We start by showing how to converge to an optimal solution when an initial feasible solution is given. To compute the initial solution, we also apply this algorithm on a slightly modified instance. The approach resembles the procedure in previous literature, although we apply the results from the previous section to speed up the computation of augmenting steps.

Let x be a feasible solution for the n-fold ILP, in particular $\mathcal{A}x = b$. Let x^* be an optimal one. Theorem 17 states that we can decompose the difference vector $x' = x^* - x$ into at most $2nt-1$ weighted Graver basis elements, that is

$$x' = x^* - x = \sum_{j=1}^{2nt-1} \lambda_j g_j.$$

For intuition, consider the following simple approach (this is similar to the algorithm by Hemmecke et al. [33]). Suppose we are able to guess the best vector $\lambda_i g_i = \operatorname{argmax}_j\{c^T(\lambda_j g_j)\}$ regarding the gain for the objective function. This pair of step length λ_i and Graver element g_i is called the Graver best step. Then we can augment the current solution x by adding $\lambda_i g_i$ to it, i.e., we set $x \leftarrow x + \lambda_i g_i$. Feasibility follows because all g_j are sign-compatible. This procedure is repeated until no improving step is possible and therefore x must be optimal. In each iteration this decreases the gap to the optimal

solution by a factor of at least $1-1/(2nt)$ by the pigeonhole principle. It may be costly to guess the precise Graver best step, but for our purposes it will suffice to find an augmenting step that is approximately as good.

We will now describe how to guess λ_i. Since $x+\lambda_i g_i$ is feasible, we have that $\lambda_i g_i \leq u-x \leq u-\ell$ and $\lambda_i g_i \geq \ell - x \geq \ell - u$. Let $(g_i)_j \in \operatorname{supp}(g_i)$ be some non-zero variable. If $(g_i)_j > 0$, then $\lambda_i \leq (\lambda_i g_i)_j \leq u_j - \ell_j$. Otherwise, $(g_i)_j < 0$ and $\lambda_i \leq -(\lambda_i g_i)_j \leq -(\ell_j - u_j) = u_j - \ell_j$. Hence, it suffices to check all values in the range $\{1,\dots,\Gamma\}$, where $\Gamma = \max_j\{u_j-\ell_j\}$. Proceeding like in [29], we lower the time a bit further by not taking every value into consideration. Instead, we look at guesses of the form $\lambda' = 2^k$ for $k \in \{0,\dots,\lfloor\log(\Gamma)\rfloor\}$. Doing so we lose a factor of at most 2 regarding the improvement of the objective function, since $c^T(\lambda' g_i) > 0.5 \cdot c^T(\lambda_i g_i)$ when taking $\lambda' = 2^{\lfloor\log(\lambda_i)\rfloor} > \lambda_i/2$. Fix λ' to the value above. Next we describe how to compute an augmenting step that is at least as good as $\lambda' g_i$. Note that g_i is a solution of

$$\begin{aligned} \mathcal{A}y &= 0 \\ \|y\|_1 &\leq k \\ \lceil\frac{\ell-x}{\lambda'}\rceil \leq y &\leq \lfloor\frac{u-x}{\lambda'}\rfloor, \end{aligned}$$

where $k = O(rs\Delta)^{rs}$ is the bound on the norm of Graver elements from Theorem 18. Suppose we have guessed some partition $P = \{P_1,\dots,P_{k^2}\}$ of the bricks such that of each P_j only a single brick has non-zero variables in g_i. Clearly, the augmenting step $\lambda' y^*$, where y^* is a (P,k)-best step with bounds $\overline{\ell} = \lceil\frac{\ell-x}{\lambda'}\rceil$ and $\overline{u} = \lfloor\frac{u-x}{\lambda'}\rfloor$ would be at least as good as $\lambda' g_i$. Indeed Lemma 21 explains how to compute such a (P,k)-best step dynamically and when we add $\lambda' g_i$ to x we only change the bounds of at most k^3 many variables. Hence, it is very efficient to recompute (P,k)-best steps until we have converged to the optimal solution. However, valid choices of λ' and P might be different in every iteration. Regarding λ', we simply compute (P,k)-best steps for each of the $O(\log(\Gamma))$ many guesses and take the best among them. We proceed similarly for P. We guess a small number of partitions and guarantee that always at least one of them is valid. For this purpose we employ splitters. More precisely, we compute a (n,k,k^2) splitter of the n bricks. Since g_i has a norm bounded by k, it can also only use at most k bricks. Therefore the splitter always contains a partition $P = \{P_1,\dots,P_{k^2}\}$ where g_i only uses a single brick in every P_j.

To recap, in every iteration we solve a (P,k)-best step problem for every guess λ' and every partition P in the splitter and take the overall best solution as an improving direction $\lambda' y^*$. Then we update our solution x by adding $\lambda' y^*$ onto it. At most k^2 many bricks change (and within each brick only k variables can change) and therefore we can efficiently recompute the (P,k)-best steps for every guess for the next iteration. This way we guarantee that we improve the solution by a factor of at least $1-1/(4nt)$ in every iteration.

Recall that we still have to find an initial solution. This solution can indeed be computed by using the augmenting step algorithm described above. To

obtain an initial solution for our n-fold ILP, we construct a new n-fold ILP which has a trivial feasible solution and whose optimal solution corresponds to a feasible solution of the original problem.

First we extend our n-fold matrix $\mathcal{A}$ by adding $(r+s)n$ new columns as follows: After the first block $(A_1, B_1, 0, \dots, 0)^T$ add $r+s$ columns. The first r ones will contain an $r \times r$ identity matrix we call $\mathcal{I}_r$. This matrix $\mathcal{I}_r$ has all ones in the diagonal. All other entries are zero. The next s columns will contain an $s \times s$ identity matrix $\mathcal{I}_s$. This submatrix will start at row $r+1$. Again all other entries are zeros in these columns. After the next block we again introduce $r+s$ new columns, the first r ones containing just zeros, the next an $\mathcal{I}_s$ matrix at the height of B_2. We repeat this procedure of adding $r+s$ columns after each block, the first r having solely zero entries and the next s containing $\mathcal{I}_s$ at the height of B_i until our resulting matrix $\mathcal{A}^{\text{init}}$ for finding the initial solution looks like the following:

$$\mathcal{A}^{\text{init}} = \begin{pmatrix} A_1 & \mathcal{I}_r & 0 & A_2 & 0 & 0 & \dots & A_n & 0 & 0 \\ B_1 & 0 & \mathcal{I}_s & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & B_2 & 0 & \mathcal{I}_s & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & & B_n & 0 & \mathcal{I}_s \end{pmatrix}.$$

Due to our careful extension $\mathcal{A}^{\text{init}}$ has again n-fold structure. For clarity the relevant submatrices are framed in the matrix above. Remark that zero entries inside of a block do not harm as solely the zeros outside of the blocks are necessary for an n-fold structure. At first glance, it seems that for the right-hand side b we now have a trivial solution consisting only of the new columns. Keep in mind, however, that the old variables have upper and lower bounds and that 0 might be outside these bounds. In order to handle this case we subtract ℓ, the lower bound, from all upper and lower bounds and set the right-hand side to $b' = b - \mathcal{A}\ell$. We get an equivalent n-fold ILP where every solution is shifted by ℓ. Now we can find a feasible solution (for b') using solely the new variables by defining

$$y' = (0, \dots, 0, b'_1, b'_2, \dots b'_{r+s}, 0, \dots, 0, \\ b'_{r+s+1}, \dots, b'_{r+2s}, 0, \dots, 0, b'_{r+ns-s}, b'_{r+ns})^T$$

where each non-zero entry corresponds to the columns containing the submatrices $\mathcal{I}_r$ and $\mathcal{I}_s$ respectively with a multiplicity of the remaining right-hand side b'. Next we introduce an objective function that penalizes using the new columns by having non-zero entries c'_i corresponding to the positions of the new variables. We set

$$c^{\text{init}} = (0, \dots 0, c'_1, \dots c'_{r+s}, 0, \dots 0, c'_{r+ns-s}, \dots, c'_{r+ns})^T,$$

where the zero entries correspond to old variables. The values c'_i and the lower and upper bounds for the new variables depend on the sign of the right-hand side.

- If $b'_i \geq 0$, then set $c'_i = -1$, the lower bound to 0, and the upper bound to b'_i. This way the variable can only be non-negative.
- If $b'_i < 0$, set $c_i = 1$, the lower bound to b'_i and the upper bound to 0. Hence this variable must be non-positive.

Clearly a solution has a value of 0, if and only if none of the new columns are used and no solution of better value is possible. Hence, if we use our augmenting step algorithm and solve this problem optimally, we either find a solution with value 0 or one with a negative value. In the former, we indeed have not taken any of the new columns into our solution, therefore we can delete the new columns and obtain a solution for the original problem (after adding ℓ to it). Otherwise, there is no feasible solution for the original problem as we solved the problem optimal regarding the objective function.

Theorem 22. *The dynamic augmenting step algorithm described above computes an optimal solution for the n-fold Integer Linear Program problem in time* $(rs\Delta)^{O(r^2s+s^2)} \cdot O(L^2 \cdot nt \log^5(nt))$ *when finite variable bounds are given for each variable. Here L is the encoding length of the largest occurring number in the input.*

Proof. Due to Theorem 17 we know that the difference vector of an optimal solution x^* to our current solution x, i.e. $x' = x^* - x$, can be decomposed into $2nt - 1$ weighted Graver basis elements. Hence, if we adjust our solution x with the Graver best step, we reduce the gap between the value of an optimal solution and our current solution by a factor of at least $1 - 1/(2nt)$ due to the pigeonhole principle. Our algorithm finds an augmenting step that is at least half as good as the Graver best step. Therefore, the gap to the optimal solution is still reduced by at least a factor of $1 - 1/(4nt)$.

Regarding the running time we first have to compute the splitter. Theorem 19 says, that this can be done in time $k^{O(1)} \cdot n \log(n) = (rs\Delta)^{O(rs)} \cdot n \log(n)$. Next we have to try all values for the weight λ. Due to our step-length we get $O(\log(\Gamma))$ guesses. Recall that Γ denotes the largest difference between an upper bound and the corresponding lower bound, i.e., $\Gamma = \max_j\{u_j - \ell_j\}$. Fixing one, we have to find the best improving direction regarding each of the $((rs\Delta)^{O(rs)})^{O(1)} \log(n) = (rs\Delta)^{O(rs)} \log(n)$ partitions. In the first iteration we have to set up the tables in time $k^{O(r)} \cdot \Delta^{O(r^2+s^2)} \cdot nt = (rs\Delta)^{O(r^2s)} \cdot \Delta^{O(r^2+s^2)} \cdot nt$ by computing the gain for each possible summand for each set and setting up the data structure. In each following iteration we update each table and search for the optimum in time $k^{O(r)} \cdot \Delta^{O(r^2+s^2)} \cdot \log(nt) = (rs\Delta)^{O(r^2s)} \cdot \Delta^{O(r^2+s^2)} \cdot \log(nt)$. Now it remains to bound the number I of iterations needed to converge to an optimal solution. To obtain such a bound we calculate:

$$1 > (1 - 1/(4nt))^I |c^T(x^* - x)|.$$

By reordering the term, we get

$$I < \frac{-\log(|c^T(x^* - x)|)}{\log(1 - 1/(4nt))}.$$

As $\log(1+x) = \Theta(x)$, we can bound $\log(1 - 1/(4nt))$ by $\Theta(-1/(4nt))$ and thus

$$I < O(\frac{-\log(|c^T(x^* - x)|)}{-1/(4nt)}) \leq O(4nt\log(|c^T(x^* - x)|)).$$

As the maximal difference between the current solution x and an optimal one x^* can be at most the maximal value of c times the largest number in between the bounds for each variable, we get $|c^T(x^* - x)| \leq nt\max_i |c_i| \cdot \Gamma$ and thus

$$\begin{aligned} I &< O(4nt\log(|c^T(x^* - x)|)) \\ &\leq O(nt\log(nt\max_i |c_i| \cdot \Gamma)) \leq O(nt\log(nt\Gamma\max_i |c_i|)). \end{aligned}$$

Let L denote the encoding length of largest integer in the input. Clearly 2^L bounds the largest absolute value in c and thus we get

$$I < O(nt\log(nt\Gamma\max_i |c_i|)) = O(nt\log(nt\Gamma 2^L)).$$

Hence after this amount of steps by always improving the gain by a factor of at least $1 - 1/(4nt)$ we close the gap between the initial solution and an optimal one. Given this, we can now bound the overall running time with:

$$\begin{aligned} &\underbrace{(rs\Delta)^{O(rs)} \cdot n\log(n)}_{\text{Splitter}} \\ &+ \underbrace{(rs\Delta)^{O(rs)}\log(n)}_{\text{Partitions}} \cdot \underbrace{(rs\Delta)^{O(r^2s)} \cdot (rs\Delta)^{O(r^2+s^2)} \cdot nt}_{\text{First Iteration}} \\ &+ \underbrace{O(nt\log(nt\Gamma 2^L))}_{I} \cdot \underbrace{O(\log(\Gamma))}_{\lambda \text{ Guesses}} \cdot \underbrace{(rs\Delta)^{O(rs)}\log(n)}_{\text{Partitions}} \\ &\quad \cdot \underbrace{(rs\Delta)^{O(r^2s)} \cdot (rs\Delta)^{O(r^2+s^2)} \cdot \log(nt))}_{\text{Update Time}} \\ &= O((nt\log(nt\Gamma 2^L)) \cdot O(\log(\Gamma)) \cdot (rs\Delta)^{O(r^2s+s^2)} \cdot \log^2(nt) \\ &= (rs\Delta)^{O(r^2s+s^2)} \cdot O(\log^2(\Gamma + 2^L) \cdot nt\log^3(nt)). \end{aligned}$$

Here *Splitter* denotes the time to compute the initial set $\mathcal{P}$ of partitions and *Partitions* denotes the cardinality of $\mathcal{P}$. *First Iteration* is the time to solve the first iteration of the (P,k)-best step problem. Further λ *Guesses* is the number of guesses we have to do to get the right weight and lastly *Update Time* is the time needed to solve each following (P,k)-best step including updating the bounds and data structures.

Note, that we still have to argue about finding the initial solution, since in the construction of the modified n-fold ILP the parameters slightly change. The length of a brick expand to $t' = t + r + s$. This, however, can be hidden in the O-Notation of $(rs\Delta)^{O(r^2s+s^2)}$. Further, Γ', the biggest difference in upper and lower bounds can change as we introduced new variables admitting new bounds. The difference between the bounds of old variables does not change. For the new variables, however, the difference can be as large as $\|b'\|_\infty$. Thus we bound this value by

$$\|b'\|_\infty = \|b - \mathcal{A}\ell\|_\infty \leq \|b\|_\infty + \|\mathcal{A}\ell\|_\infty \leq \|b\|_\infty + \Delta \cdot \|\ell\|_1 \leq O(\Delta \cdot nt \cdot 2^L).$$

We conclude that the running time for finding an initial solution (and also the overall running time) is

$$\begin{aligned}
&(rs\Delta)^{O(r^2s+s^2)} O(\log^2(\Gamma' + 2^L) nt \log^3(nt)) \\
&= (rs\Delta)^{O(r^2s+s^2)} O(\log^2(\Delta 2^L nt) nt \log^3(nt)) \\
&= (rs\Delta)^{O(r^2s+s^2)} \cdot L^2 nt \log^5(nt). \quad \square
\end{aligned}$$

HANDLING INFINITE BOUNDS Remark, that if no finite bounds are given for all variables, we have to introduce some artificial bounds first. Here we can proceed as in [29], where first the LP relaxation is solved to obtain an optimal fractional solution z^*. Using the proximity results from [29], we know that an optimal integral solution x^* exists such that $\|x^* - z^*\|_1 \leq nt(rs\Delta)^{O(rs)}$. This allows us to introduce artificial upper bounds for the unbounded variables. Remark that this comes at the price of solving the corresponding relaxation of the n-fold Integer Linear Program problem. However we also lessen the dependency from L^2 to L as the finite upper and lower bounds can also be bounded more strictly due to the same proximity result. This yields an overall running time of $(rs\Delta)^{O(r^2s+s^2)} \cdot L \cdot nt \log^5(nt) + \text{LP}$. Nevertheless, solving this LP can be very costly, indeed it is not clear if a potential algorithm even runs in time linear in n. Thus, it may even dominate the running time of solving the n-fold ILP with finite upper bounds. Fortunately we can circumvent the necessity of solving the LP as we will describe in the following section using new structural results.

Theorem 23. *The dynamic augmenting step algorithm described above computes an optimal solution for the n-fold Integer Linear Program problem in time* $(rs\Delta)^{O(r^2s+s^2)} \cdot L \cdot nt \log^5(nt) + LP$ *when some variables have infinite upper bounds. Here LP is the running time to solve the corresponding relaxation of the n-fold ILP problem.*

4.4 STRUCTURAL PROPERTIES OF SOLUTIONS

In the following, we state that even with infinite variable bounds in an n-fold ILP there always exists a solution of small norm (if the n-fold ILP has a finite optimum). Therefore, we can apply the algorithm for finite variable bounds by replacing every infinite one with this value.

Lemma 24. *If the n-fold ILP is feasible and $\overline{y}$ is some vector satisfying the variable bounds, then there exists a feasible solution x with $\|x\|_1 \leq O(rs\Delta)^{rs+1} \cdot (\|\overline{y}\|_1 + \|b\|_1)$*

Proof. We take the same construction as in the algorithm for finding a feasible solution in. Indeed, this construction was not setup for infinite bounds, but we consider the straight-forward adaption where infinite bounds simply stay the same. The useful property is that an optimal solution for this n-fold ILP is a feasible solution for the original one. Recall, the construction has a right-hand side b' with $\|b'\|_1 \leq \|\mathcal{A}\overline{y}\|_1 + \|b\|_1$, the value of t becomes $t' = t + r + s$, and the objective function c' consists only of the values $\{-1, 0, 1\}$. Moreover, there is a feasible solution y with $\|y\|_1 = \|b'\|_1$. Let x^* be an optimal solution for this altered n-fold ILP that minimizes $\|x^* - y\|_1$. We consider the decomposition into Graver elements $\sum_{i=1}^{2n(t+r+s)-1} \lambda_i g_i = x^* - y$. Then $c'^T g_i > 0$ or $\lambda_i = 0$ for all i, since otherwise $x^* - g_i$ would be either a better solution than x^* or equally good, but closer. It follows that $c'^T g_i \geq 1$ by discreteness of c. Also, by Theorem 18, $\|g_i\|_1 \leq O(rs\Delta)^{rs}$. Recall that by construction $c'^T x^* = 0$ and $c'^T y = -\|b'\|_1$, which implies $\sum_i \lambda_i \leq \|b'\|_1$. Therefore,

$$\begin{aligned}\|x^*\|_1 &\leq \|y\|_1 + \|\sum_i \lambda_i g_i\|_1 \leq \|b'\|_1 + \sum_i \lambda_i \|g_i\|_1 \\ &\leq \|b'\|_1 + \|b'\|_1 \cdot O(rs\Delta)^{rs} \leq (\|b\|_1 + \|\overline{y}\|_1) \cdot O(rs\Delta)^{rs+1}.\end{aligned}$$

Here we use that $\|b'\|_1 \leq \|b\|_1 + \|\mathcal{A}\overline{y}\|_1 \leq \|b\|_1 + (r+s)\Delta \cdot \|\overline{y}\|_1$. □

Lemma 25. *If the n-fold ILP is bounded and feasible, then there exists an optimal solution x with $\|x\|_1 \leq (rs\Delta)^{O(rs)} \cdot (\|b\|_1 + nt\zeta)$, where ζ denotes the largest absolute value among all finite variable bounds.*

Proof. Clearly there exists a (possibly infeasible) solution $\overline{y}$ satisfying the bounds with $\|\overline{y}\|_1 \leq nt\zeta$. By the previous lemma we know that there is a feasible solution y with $\|y\|_1 \leq (rs\Delta)^{O(rs)} \cdot (\|b\|_1 + nt\zeta)$. Let x^* be an optimal solution of minimal norm. W.l.o.g. assume that $x^* - y$ has only non-negative entries. If there is a negative entry, consider the equivalent n-fold problem with the corresponding column inverted and its bounds inverted and swapped.

We know that there is a decomposition of $x^* - y$ into weighted Graver basis elements $\sum_{i=1}^{2nt-1} \lambda_i g_i = x^* - y$. Since every g_i is sign-compatible with $x^* - y$, we have that all g_i are non-negative as well. Furthermore, it holds that $c^T g_i > 0$ or $\lambda_i = 0$ for every g_i, since otherwise $x^* - g_i$ would be a solution of smaller norm with an objective value that is not worse. Now suppose toward contradiction that there is some g_i where all variables in $\operatorname{supp}(g_i)$ have infinite upper bounds. Then the n-fold ILP is clearly unbounded, since $y + \alpha \cdot g_i$ is feasible for every $\alpha > 0$ and in this way we can scale the objective value beyond any bound. Thus, every Graver basis element adds at least the value 1 to some finitely bounded variable. This implies that $\sum_i \lambda_i \leq \|y\|_1 + nt\zeta$: If

not, then by pigeonhole principle there is some finitely bounded variable x^*_j with

$$x^*_j = y_j + (\sum_i \lambda_i g_i)_j > y_j + \zeta + |y_j| \geq \zeta.$$

Since x^* is feasible, this cannot be the case. We conclude,

$$\begin{aligned}\|x^*\|_1 &\leq \|y\|_1 + \sum_i \lambda_i \|g_i\|_1 \leq \|y\|_1 + O(rs\Delta)^{rs} \cdot \sum_i \lambda_i \\ &\leq O(rs\Delta)^{rs} \cdot (\|y\|_1 + nt\zeta) \leq (rs\Delta)^{O(rs)} \cdot (\|b\|_1 + nt\zeta). \quad \square\end{aligned}$$

This yields an alternative approach to solving the LP relaxation, because now we can simply replace all infinite bounds with $\pm(rs\Delta)^{O(rs)} \cdot nt \cdot 2^L$. Then we can apply the algorithm that works only on finite variable bounds. The new encoding length L' of the largest integer in the input can be bounded by

$$L' \leq \log((rs\Delta)^{O(rs)} \cdot 2^L \cdot nt) \leq O(rs \cdot \log(rs\Delta) \cdot L \cdot \log(nt)).$$

This way we obtain the following.

Theorem 26. *We can compute an optimal solution for an n-fold ILP in time $(rs\Delta)^{O(r^2s+s^2)} \cdot L^2 \cdot nt \log^7(nt)$.*

In a similar way, we can derive the following bound on the sensitivity of an n-fold ILP. This bound is not needed in our algorithm, but may be of independent interest, since it implies small sensitivity for problems that can be expressed as n-fold ILPs.

Theorem 27. *Let x be an optimal solution of an n-fold ILP with right-hand side b, in particular, $\mathcal{A}x = b$. If the right hand side changes to b' and the n-fold ILP still has a finite optimum, then there exists an optimal solution x' for b' ($\mathcal{A}x' = b'$) with $\|x - x'\|_1 \leq O(rs\Delta)^{rs} \cdot \|b - b'\|_1$.*

It is notable that this bound does not depend on n. This is in contrast to the known bounds for the distance between LP and ILP solutions of an n-fold ILP [29].

Proof. Consider the matrix $\mathcal{A}^{\text{init}}$ from the construction used for finding an initial solution, that is, identity matrices are added after every block. As opposed to the proof of Lemma 24, we leave everything except for the matrix the same. In particular, we do not change the value in the objective function c and new columns get a value of 0. As the right-hand side of the n-fold ILP we use b'. For some solution x, we write x_{old} and x_{new} for the vector restricted to the old variables (with all others 0) and the variables added in the matrix $\mathcal{A}^{\text{init}}$, respectively. This means $x = x_{\text{old}} + x_{\text{new}}$.

Let x be an optimal solution with $\mathcal{A}^{\text{init}} \cdot x_{\text{new}} = b' - b$ and x' one with $\mathcal{A}^{\text{init}} \cdot x'_{\text{new}} = 0$. Here we assume that x' is chosen so as to minimize $\|x - x'\|_1$. Those solutions naturally correspond to solutions of the original n-fold ILP with right-hand side $b = b' - (\mathcal{A}^{\text{init}} \cdot x_{\text{new}})$ and $b' = b' - \mathcal{A}^{\text{init}} \cdot x'_{\text{new}}$. Let

$\sum_{i=1}^{2n(t+r+s)-1} \lambda_i g_i = x' - x$ be the decomposition into Graver basis elements. Suppose toward contradiction there is some g_i where all of $\mathrm{supp}(g_i)$ are old variables. If $c^T g_i > 0$, then x is not optimal, because $x + g_i$ is feasible and has a better objective value. If on the other hand $c^T g_i \leq 0$, then $x' - g_i$ is a solution of at least the same value as x' and thus $\|x - x'\|_1$ is not minimal. Indeed, this means $\|(g_i)_{\text{new}}\|_1 \geq 1$ for all g_i. In other words, each graver element contains a non-zero new variable. Recall that $\mathcal{A}^{\text{init}}$ is the identity matrix when restricted to the new variables (plus some zero columns). Due to the sign-compatibility we get

$$\sum_i \lambda_i \leq \|(\sum_i \lambda_i g_i)_{\text{new}}\|_1 = \|\mathcal{A}^{\text{init}} \cdot (\sum_i \lambda_i g_i)_{\text{new}}\|_1 = \|\mathcal{A}^{\text{init}} \cdot (x' - x)_{\text{new}}\|_1 = \|b - b'\|_1.$$

We conclude,

$$\|x - x'\|_1 = \|\sum_i \lambda_i g_i\|_1 \leq O(rs\Delta)^{rs} \cdot \sum_i \lambda_i \leq O(rs\Delta)^{rs} \cdot \|b - b'\|_1. \quad \square$$

Proof. The difference vector $b^* = b' - b$ will have all entries equal zero except one with value 1. The position will either correspond to the A_1 blocks or to one of the A_2 blocks. Now extent the n-fold A as follows: Add one column after each brick. Each of the new columns will solely consist of zero entries, except of one column $j = b^*$. This column will be placed either after the first brick, if the 1 entry corresponds to the A_1 block or after the corresponding A_2 block. Clearly this new matrix A^* is an n-fold. Further a solution x^* for $A^* x^* = b'$ clearly matches all entries of x for the corresponding columns and set the position j to 1, the remaining new columns equal zero.

The next step is to bound the distance of a solution x' to x using x^*. Therefore look at the vector $x^* - x' = x'^*$. The vector x'^* clearly is an element of the kern of A^*. Thus is can be decomposed into its Graver basis elements $g_1, \ldots, g_k$ with each Graver element being bounded by $O((rs\Delta)^{rs})$. Each of the elements are sign-compatible, w.l.o.g. assume all entries are positive. Then clearly only one element g_z contains a 1 in the jth column. For this circle g_z set the one entry to zero, such that we obtain a new vector g_z^*. This is clearly not an element of the kern of A^* anymore, since $A^* g_z^* = (0 \ldots 010 \ldots 0)^T$, where the non zero entry appears in the jth place. Hence we can simply add g_z^* to our solution x, such that $A(x + g_z^*) = A(x') = b'$. Indeed to legitimate add these vectors, we first have to drop all additional columns of g_z^*, which where all zero and thus did not influence the right-hand side. Since g_z is a Graver basis element and g_z^* is obtained from it by just setting one entry to zero, we can bound g_z^* with $O((rs\Delta)^{rs})$. Hence there exists a solution x' close to x.

Indeed we also have to prove the optimality of this solution x'. Therefore look at the remaining Graver basis elements $g_1, \ldots, g_k$ without g_z. If a cycle is of positive gain regarding the objective function c, then is could have been added to x and x was thus not an optimal solution. On the other hand if the gain is negative, then we could have subtracted it from x' and thus x'

would not have been optimal. Hence the gain for each Graver basis equals zero and thus does not contribute to a more optimal solution. Hence $x + g_z^*$ is optimal. □

5 THE SANTA CLAUS PROBLEM

The following problem is notable for its influencial proof technique that has inspired also the work in the later chapters.

A generalization of the problem we consider goes back to Bansal and Srividenko [12]. In the Santa Claus problem there are players $\mathcal{P}$ and resources $\mathcal{R}$. Every resource j has a value $v_{ij} \geq 0$ for player i. The goal is to find an allocation $\sigma : \mathcal{R} \to \mathcal{P}$ such that $\min_{i \in \mathcal{P}} \sum_{j \in \sigma^{-1}(i)} v_{ij}$ is maximized.

In the restricted variant, we consider only values $v_{ij} \in \{0, v_j\}$ where $v_j > 0$ is a value depending only on the resource. This can also be seen as each player desiring a subset $\mathcal{R}(i)$ of resources which have a value of v_j for him, whereas other resources cannot be assigned to him.

For the restricted Santa Claus problem there is a strong LP relaxation, the configuration LP. The proof that this has a small integrality gap (see [8]) is not trivial. It works by defining an exponential time local search algorithm which is guaranteed to return an integral solution of value not much less than the fractional optimum. This technique has since been used in other problems, like the minimization of the makespan [68, 41]. Significant research has also gone into making the proof constructive [62, 7, 40]. Yet, no improvement of the bound of 4 on the integrality gap has been found. We show that the original analysis is not tight and can be improved to $3 + 5/6 \approx 3.8333$ [1]. Although the proof of the integrality gap is not constructive, it does allow to estimate the optimum up to a factor of 3.8334 in polynomial time by solving the configuration LP. Furthermore, this provides strong evidence for the existence of a constructive algorithm with approximation ratio better than 4.

Theorem 28. *The integrality gap of the configuration LP for the restricted Santa Claus problem is at most* $3 + 5/6$.

CONFIGURATION LP. The configuration LP is an exponential size LP relaxation for the Santa Claus problem, but its solution can be approximated in polynomial time within a ratio of $(1 + \epsilon)$ for every $\epsilon > 0$ [12]. For every player i and every $\tau \geq 0$ let

$$\mathcal{C}(i, \tau) = \left\{ S \subseteq \mathcal{R}(i) : \sum_{j \in S} v_j \geq \tau \right\}.$$

1 The same bound was found simultaneously and independently by Cheng and Mao [22]. Later the same authors improved it to 3.808 [23].

These are the configurations for player i and value τ. They are a selection of resources that have value at least τ and are desired by player i. The optimum OPT^* of the configuration LP is the highest τ such that the following linear program is feasible.

Primal of the configuration LP for restricted Santa Claus

$$\sum_{C \in \mathcal{C}(i,\tau)} x_{i,C} \geq 1 \qquad \forall i \in \mathcal{P}$$

$$\sum_{i \in \mathcal{P}} \sum_{C \in \mathcal{C}(i,\tau) : j \in C} x_{i,C} \leq 1 \qquad \forall j \in \mathcal{R}$$

$$x_{i,C} \geq 0$$

This linear program assigns at least one configuration to every player and makes sure that no resource is taken more than once. We will also consider the dual of this LP, which is constructed after adding the neglectable objective function $\min(0,\ldots,0) \cdot x$:

Dual of the configuration LP for restricted Santa Claus

$$\max \sum_{i \in \mathcal{P}} y_i - \sum_{j \in \mathcal{R}} z_j$$

$$\sum_{j \in C} z_j \geq y_i \qquad \forall i \in \mathcal{P}, C \in \mathcal{C}(i,\tau)$$

$$y_i, z_j \geq 0$$

From duality we can derive the following condition.

Lemma 29. *Let $y \in \mathbb{R}^{\mathcal{P}}_{\geq 0}$ and $z \in \mathbb{R}^{\mathcal{R}}_{\geq 0}$ such that $\sum_{i \in \mathcal{P}} y_i > \sum_{j \in \mathcal{R}} z_j$ and for every $i \in \mathcal{P}$ and $C \in \mathcal{C}(i,\tau)$ it holds that $\sum_{j \in C} z_j \geq y_i$, then $\mathrm{OPT}^* < \tau$.*

It is easy to see that if such a solution y, z exists, then every component can be scaled by a constant to obtain a feasible solution greater than any given value. Hence, the dual must be unbounded and therefore the primal must be infeasible.

NOTATION AND PREPROCESSING. We denote by $\alpha = 3 + 5/6$ the bound on integrality we want to prove. We write $v(S) = \sum_{j \in S} v_j$ for some set of resources $S \subseteq \mathcal{R}$ and use this notation also for other variables indexed by resources.

We model our problem as a hypergraph matching problem: There are vertices for all players and all resources and the hyperedges $\mathcal{H}$ each consist of exactly one player i and a set of resources $C \subseteq \mathcal{R}(i)$ where $v(C) \geq \mathrm{OPT}^*/\alpha$. However, we restrict $\mathcal{H}$ to edges that are minimal, that is to say, $v(C') < \mathrm{OPT}^*/\alpha$ for all $C' \subsetneq C$. Clearly, a matching (a set of non-overlapping edges) such that every player is in one matching edge corresponds to a solution of value OPT^*/α. For a set of edges F we write $F_{\mathcal{P}}$ as the set of players in these

edges and $F_{\mathcal{R}}$ as the resources in the edges. Moreover, for a single edge e we simplify the notation to $e_{\mathcal{R}}$ and $e_{\mathcal{P}}$ for $\{e\}_{\mathcal{R}}$ and $\{e\}_{\mathcal{P}}$.

We reduce our problem to extending an existing matching: The algorithm takes an (incomplete) matching M and a player i_0, which is not in M. Then it extends the matching to also cover i_0. Starting with the empty matching and calling the algorithm for each player yields the desired matching. No state except for the matching is saved between the iterations.

5.1 ALGORITHM

We consider the local search algorithm from [8]. It is the same algorithm with a slightly different presentation that is inspired by [62].

Two types of edges play a crucial role in the algorithm: A sequence of edges $e_1^A, \dots, e_\ell^A \in \mathcal{H} \setminus M$ (the addable edges) and sets $B_M(e_1^A)$, $B_M(e_2^A)$, $\dots$, $B_M(e_\ell^A) \subseteq M$ (the blocking edges for each addable edge). Note that the order of the addable edges is important. An addable edge is an edge that the algorithm hopes to add to M — either to cover the new player i_0 or to free a player of a blocking edge. A blocking edge is an edge in M that conflicts with an addable edge, i.e., that overlaps with an addable edge on the resource part. For each addable edge e_k^A we define the blocking edges $B_M(e_k^A)$ as $\{e' \in M : e'_{\mathcal{R}} \cap (e_k^A)_{\mathcal{R}} \neq \emptyset\}$. We write $B_M(A)$ for $\bigcup_{e \in A} B_M(e)$ for a set of addable edges A.

The algorithm (see Alg. 1) runs a main loop until the matching has been extended. In each iteration it first adds a new addable edge that does not overlap in resources with any existing addable or blocking edge, but contains a player that is either i_0 or the player in an existing blocking edge. Then it consecutively swaps addable edges that are not blocked for the blocking edge they are supposed to free. Also, all the addable and blocking edges added at a later time are removed, since they might be obsolete now.

5.2 ANALYSIS

We start by mentioning some basic properties of the algorithm.

Fact 30. *The swap in the inner loop does not create new blocking edges and removes a blocking edge from e_k^A.*

We refer to the edges e_k^A, e', and e_ℓ^A as in algorithm. The addable edges do not overlap on the resource part. Since e_ℓ^A was an addable edge and is turned into a matching edge, it cannot become a blocking edge of an existing addable edge. e' was a blocking edge of e_k^A, but is removed from the matching. Hence, it is no longer a blocking edge.

Fact 31. *The blocking edges $B_M(e_k^A)$ for every addable edge e_k^A are pairwise disjoint.*

Let $k < k'$ and consider the blocking edges $B_M(e_k^A)$ and $B_M(e_{k'}^A)$. After e_k^A is added, the algorithm never adds a new blocking edge to it (see previous

Input: Partial matching M and unmatched player i_0
Result: Partial matching M' with $M'_{\mathcal{P}} = M_{\mathcal{P}} \cup \{i_0\}$
$\ell \leftarrow 0$;
loop
  $A \leftarrow \{e_1^A, \dots, e_\ell^A\}$; // initially empty
  $\ell \leftarrow \ell + 1$;
  let $e_\ell^A \in \mathcal{H} \setminus M$ with $(e_\ell^A)_{\mathcal{R}} \cap (A \cup B_M(A))_{\mathcal{R}} = \emptyset$ and $(e_\ell^A)_{\mathcal{P}} \in B_M(A)_{\mathcal{P}} \cup \{i_0\}$;
  // the existence of e_ℓ^A is proved in Lemma 35
  while $B_M(e_\ell^A) = \emptyset$ **do**
    if *there is* $k < \ell$ *and* $e' \in B_M(e_k^A)$ *with* $e'_{\mathcal{P}} = (e_\ell^A)_{\mathcal{P}}$ **then**
      // e' and e_k^A are unambiguous, since M is a matching
      $M \leftarrow M \setminus \{e'\} \cup \{e_\ell^A\}$; // swap e' for e_ℓ^A
      delete $e_{k+1}^A, \dots, e_\ell^A$;
      $\ell \leftarrow k$;
    else
      $M \leftarrow M \cup \{e_\ell^A\}$; // $(e_\ell^A)_{\mathcal{P}} = i_0$, see Fact 32
      return M ;

Algorithm 1: Local search for restricted Santa Claus

fact). Thus, the blocking edges in $B_M(e_k^A)$ already existed when $e_{k'}^A$ was added. However, $e_{k'}^A$ was chosen such that its resources are disjoint from existing blocking edges. Therefore, no blocking edge can be in both $B_M(e_k^A)$ and $B_M(e_{k'}^A)$.

Fact 32. *For every addable edge e_k^A, $(e_k^A)_{\mathcal{P}}$ is either i_0 or the player in a blocking edge in $B_M(e_{k'}^A)$ for some $k' < k$.*

When the addable edge is added, this is true by definition. We only have to look at the case, where $(e_k^A)_{\mathcal{P}}$ is the player in a blocking edge in $B_M(e_{k'}^A)$ and this blocking edge is removed by the swap operation. In this case, however, every addable edge after $e_{k'}^A$, in particular, e_k^A, is deleted.

Fact 33. *At the beginning of each iteration of the main loop, the blocking edges $B_M(e_k^A)$ for each addable edge e_k^A are non-empty.*

We suppose that at the beginning of one iteration this is true and argue that it is also true in the next iteration. After adding e_ℓ^A, this is the only addable edge that might have no blocking edges. If so, it is deleted in the inner loop (or the algorithm terminates). This removes a blocking edge from one addable edge, which is the new last edge e_ℓ^A. If it has no blocking edges, it is again deleted. This is repeated until the last edge has a blocking edge. Only then the next iteration of the main loop starts.

Lemma 34 ([8])**.** *The algorithm terminates after at most $2^{|\mathcal{P}|-1}$ many iterations of the main loop.*

Proof. Consider the signature vector

$$s(A) = (|B_M(e_1^A)|, |B_M(e_2^A)|, \ldots, |B_M(e_\ell^A)|, \infty).$$

This vector decreases lexicographically after every iteration of the main loop: If the inner loop is never executed, the first components, i.e., $|B_M(e_1^A)|$ through $|B_M(e_\ell^A)|$ remain the same. However, we replace the $(\ell+1)$'th component ∞ by some finite value $|B_M(e_{\ell+1}^A)|$, where $e_{\ell+1}^A$ is the newly selected addable edge. Hence, we can assume w.l.o.g. that the inner while loop is executed at least once. Let ℓ' be the number of addable edges after the last execution of the inner loop. Then $|B(e_{\ell'}^A)|$ has decreased by the swap operation (see Fact 30). It follows that the signature vectors in each iteration of the main loop are pairwise different.

The number of possible signature vectors can easily be bounded by $(|\mathcal{P}|+1)^{|\mathcal{P}|+1}$: All $B_M(e_k^A)$ are pairwise disjoint, non-empty, and their cardinalities sum to up at most $|M| \le |\mathcal{P}|$. Hence, the length of $s(A)$ is at most $|\mathcal{P}|+1$ and each component has one of $|\mathcal{P}|+1$ many values (including ∞). A clever idea from [8] even gives a bound of $2^{|\mathcal{P}|-1}$: We have that $\sum_{k=1}^{\ell} |B_M(e_k^A)| = |B_M(A)| \le |M| \le |\mathcal{P}|$ and $|B_M(e_k^A)| \ge 1$ for all k. There is an bijection between signature vectors and possibilities of placing separators on a line of $|\mathcal{P}|$ elements. $|B_M(e_k^A)|$ is the number of elements between the $(k-1)$-th and k-th separator. The number of possibilities of placing separators between the $|\mathcal{P}|$ elements is the number of subsets of $|\mathcal{P}|-1$ elements, i.e. $2^{|\mathcal{P}|-1}$. □

Clearly the inner loop also terminates after finitely many iterations, since in each iteration ℓ is decreased.

Lemma 35. *An edge e_ℓ^A as selected in the beginning of the main loop always exists.*

Proof. In the proof we use the constant $\beta = 1 + 8/15 \approx 1.53333$, that has been chosen so as to minimize α. Consider the beginning of an iteration of the main loop. Let $A = \{e_1^A, \ldots, e_\ell^A\}$ and assume toward contradiction that no edge $e \in \mathcal{H} \setminus M$ exists with $e_{\mathcal{R}} \cap (A \cup B_M(A))_{\mathcal{R}} = \emptyset$ and $e_{\mathcal{P}} \in B_M(A)_{\mathcal{P}} \cup \{i_0\}$. In the remainder of the proof we will write B instead of $B_M(A)$ and $B(e)$ instead of $B_M(e)$, since M and A are constant throughout the proof. Define

$$y_i = \begin{cases} 1 & \text{if } i \in B_{\mathcal{P}} \cup \{i_0\}, \\ 0 & \text{otherwise.} \end{cases}$$

$$z_j = \begin{cases} 1 & \text{if } v_j \ge \mathrm{OPT}^*/\alpha \text{ and } j \in A_{\mathcal{R}} \cup B_{\mathcal{R}}, \\ \min\{1/3, \beta \cdot v_j/\mathrm{OPT}^*\} & \text{if } v_j < \mathrm{OPT}^*/\alpha \text{ and } j \in A_{\mathcal{R}} \cup B_{\mathcal{R}}, \\ 0 & \text{otherwise.} \end{cases}$$

We refer to the resources j where $v_j \ge \mathrm{OPT}^*/\alpha$ as fat resources and to others as thin resources. Note that by the minimality property of edges in $\mathcal{H}$, each edge containing a fat resource does not contain any other resources. We

call these the fat edges. Likewise, edges that contain only thin resources are referred to as thin edges.

This definition of the variables differs from the previous one (e.g. [62]) that can be used to show a bound of 4. The essential difference is that we are capping thin edge values at $1/3$. The previous definition uses the same values except for $z_j = 4/3 \cdot v_j / OPT^*$ for thin edges $j \in A_{\mathcal{R}} \cup B_{\mathcal{R}}$ (in addition, all variables are scaled by $3/4$, which has no effect). We make use of our new choice of values is the distinction, whether all resources in an edge are relatively large or there is also a very small one. This is found near the end of the proof for Claim 37.

Claim 36. *(y,z) is a feasible solution for the dual of the configuration LP with $\tau = \mathrm{OPT}^*$.*

Claim 37. *(y,z) has a negative objective value, i.e., $\sum_{j \in \mathcal{R}} z_j < \sum_{i \in \mathcal{P}} y_i$.*

By Lemma 29 this implies that the configuration LP is infeasible for OPT^*. A contradiction. □

Proof of Claim 36. Let $i \in \mathcal{P}$ and $C \in \mathcal{C}(i, \mathrm{OPT}^*)$. We need to show that $y_i \leq z(C)$. If $y_i = 0$, this is trivial. Hence, assume that $y_i = 1$ and thus $i \in B_{\mathcal{P}} \cup \{i_0\}$. If C contains a fat resource j, then $z_j = 1$, since otherwise the hyperedge consisting of j and i could be added to the addable edges. The inequation again follows easily. We can therefore focus on the case where C consists solely of thin resources and $y_i = 1$.

Since no addable edge for i remains, $v(C \setminus (A_{\mathcal{R}} \cup B_{\mathcal{R}})) < \mathrm{OPT}^*/\alpha$. Let $S \subseteq C$ be the resources $j \in C$ which have $z_j = 1/3$.

CASE 1: $|S| = 3$. Then $z(C) \geq z(S) \geq 3 \cdot 1/3 = 1$.

CASE 2: $|S| \leq 2$. Define $C' := (C \cap (A_{\mathcal{R}} \cup B_{\mathcal{R}})) \setminus S$. Then

$$v(C') > v(C) - v(C \setminus (A_{\mathcal{R}} \cup B_{\mathcal{R}})) - v(S) \geq \mathrm{OPT}^* - \mathrm{OPT}^*/\alpha - v(S).$$

Therefore,

$$\begin{aligned} z(C) \geq 1/3 \cdot |S| + \beta/\mathrm{OPT}^* \cdot v(C') &> 1/3|S| + \beta(1 - 1/\alpha - v(S)/\mathrm{OPT}^*) \\ &\geq 1/3|S| + \beta(1 - (|S|+1)/\alpha) =: (*). \end{aligned}$$

Since $\beta/\alpha = 0.4 > 1/3$, the coefficient of $|S|$ in $(*)$ is negative and thus we can substitute $|S|$ for its upper bound, i.e., 2. By inserting the values of α and β we get,

$$(*) \geq 2/3 + \beta(1 - 3/\alpha) = 1.$$ □

Proof of Claim 37. We write in the following F^f (F^t) for the fat edges (thin edges, respectively) in a set of edges F. First note that every fat resource e with positive z value must be in a fat blocking edge: By definition we have $e \in A_{\mathcal{R}} \cup B_{\mathcal{R}}$. If e is in $A_{\mathcal{R}}$, then by minimality property of edges there is an

addable edge containing solely e. Since every addable edge has a blocking edge, e must also be in $B_{\mathcal{R}}$. Therefore

$$\sum_{j \in \mathcal{R}^f} z_j \leq |B^f|.$$

Now consider thin addable edges. Since every addable edge is blocked, we have $|B(e)| \geq 1$ for every $e \in A^t$. We now proceed to show that for every $e \in A^t$

$$z(e_{\mathcal{R}} \cup B(e)_{\mathcal{R}}) \leq |B(e)|.$$

Note that for every thin resource j, we have $z_j \leq \beta / \mathrm{OPT}^* \cdot v_j$. By the minimality property of edges in $\mathcal{H}$, it holds that $v(e_{\mathcal{R}}) \leq 2\mathrm{OPT}^*/\alpha$ (each element in $e_{\mathcal{R}}$ has value at most OPT^*/α). Also $v(e'_{\mathcal{R}} \setminus e_{\mathcal{R}}) \leq \mathrm{OPT}^*/\alpha$ for each $e' \in B(e)$, since the intersection of $e_{\mathcal{R}}$ and $e'_{\mathcal{R}}$ is non-empty. If $|B(e)| \geq 2$, this implies

$$\begin{aligned} z(e_{\mathcal{R}} \cup B(e)_{\mathcal{R}}) &\leq \beta / \mathrm{OPT}^* \cdot (v(e_{\mathcal{R}}) + v(B(e)_{\mathcal{R}} \setminus e_{\mathcal{R}})) \\ &\leq \beta \cdot (2/\alpha + |B(e)| \cdot 1/\alpha) \leq |B(e)| \cdot \beta \cdot 2/\alpha = 0.8 \cdot |B(e)| < |B(e)|. \end{aligned}$$

Assume in the following that $|B(e)| = 1$. Let $v_{\min}$ be the value of the smallest element in $e_{\mathcal{R}} \cup B(e)_{\mathcal{R}}$. Then $v(e_{\mathcal{R}} \cup B(e)_{\mathcal{R}}) \leq 2\mathrm{OPT}^*/\alpha + v_{\min}$: If the smallest element is in $e_{\mathcal{R}}$, then

$$v(e_{\mathcal{R}} \cup B(e)_{\mathcal{R}}) \leq \underbrace{\mathrm{OPT}^*/\alpha + v_{\min}}_{\geq v(e_{\mathcal{R}})} + v(B(e)_{\mathcal{R}} \setminus e_{\mathcal{R}}) \leq 2\mathrm{OPT}^*/\alpha + v_{\min}.$$

The same argument (swapping the role of e and $B(e)$) holds, if the smallest element is in $B(e)$. Therefore, if $|B(e)| = 1$ and $v_{\min} \leq 1/2 \cdot \mathrm{OPT}^*/\alpha$,

$$z(e_{\mathcal{R}} \cup B(e)_{\mathcal{R}}) \leq \beta \cdot (2/\alpha + v_{\min}) \leq \beta \cdot 5/2 \cdot 1/\alpha = 1 = |B(e)|.$$

If $|B(e)| = 1$ and $v_{\min} > 1/2 \cdot \mathrm{OPT}^*/\alpha$, then $B(e)_{\mathcal{R}} \setminus e_{\mathcal{R}}, B(e)_{\mathcal{R}} \cap e_{\mathcal{R}}$, and $e_{\mathcal{R}} \setminus B(e)_{\mathcal{R}}$ have at least one element and by the minimality property of edges at most one element. Since the z value of each thin edge is at most $1/3$,

$$\begin{aligned} z(e_{\mathcal{R}} \cup B(e)_{\mathcal{R}}) &= z(e_{\mathcal{R}} \setminus B(e)_{\mathcal{R}}) + z(e_{\mathcal{R}} \cap B(e)_{\mathcal{R}}) + z(B(e)_{\mathcal{R}} \setminus e_{\mathcal{R}}) \\ &\leq 3 \cdot 1/3 = 1 = |B(e)|. \end{aligned}$$

We conclude that

$$\begin{aligned} \sum_{i \in \mathcal{P}} y_i = |B^f| + |B^t| + 1 > |B^f| + \sum_{e \in A^t} |B(e)| \\ \geq \sum_{j \in \mathcal{R}^f} z_j + \sum_{e \in A^t} z(e_{\mathcal{R}} \cup B(e)_{\mathcal{R}}) = \sum_{j \in \mathcal{R}} z_j. \quad \square \end{aligned}$$

The problem in this chapter is closely related to the one from the previous chapter. In particular, it is about balancing loads. Here, the objective is to minimize the maximum instead of maximizing the minimum.

6 THE RESTRICTED ASSIGNMENT PROBLEM

This chapter is about a special case of the problem Scheduling on Unrelated Machines, where the goal is to compute an allocation $\sigma : \mathcal{J} \to \mathcal{M}$ of the jobs $\mathcal{J}$ to the machines $\mathcal{M}$. On machine i the job j has a processing time (size) p_{ij}. We want to minimize the makespan, which is the maximum load $\max_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_{ij}$. The classical 2-approximation by Lenstra et al. [56] is still the algorithm of choice for this problem. They also show that no approximation ratio better than $3/2$ can be found in polynomial time unless $\mathrm{P} = \mathrm{NP}$. Schuurman and Woeginger [64] name it as one of the most important open questions in scheduling to close this gap.

While the general problem remains unclear, there has been progress on a special case called Restricted Assignment. Here each job j has a processing time p_j, which is independent from the machines, and a set of feasible machines $\Gamma(j)$. This means j can only be assigned to one of the machines in $\Gamma(j)$. Note that this is equivalent to the previous problem when $p_{ij} \in \{p_j, \infty\}$. The lower bound of $3/2$ holds also in the restricted case and even if given quasi-polynomial running time no better approximation ratio can be obtained, unless $\mathrm{DTIME}(2^{\mathrm{polylog}(n)}) = \mathrm{NP}$, which would contradict popular conjectures such as the Exponential Time Hypothesis.

In a seminal work, Svensson [68] proved that the configuration LP, a natural linear programming relaxation, has an integrality gap of at most $33/17$. By approximating the optimum of the configuration LP this yields an $(33/17 + \epsilon)$-estimation algorithm. However, this proof is non-constructive and no polynomial algorithm is known that can produce a solution of this quality.

For instances with only two processing times additional progress has been made. Chakrabarty et al. gave a polynomial $(2 - \delta)$-approximation for a very small δ [19]. Annamalai improved this with a $(17/9 + \epsilon)$-approximation for every $\epsilon > 0$ [6]. For this special case it was also shown that the integrality gap is at most $5/3$ [38].

In [68] and [38] the critical idea is to design a local search algorithm, which is then shown to produce good solutions. However, the algorithm has a potentially high running time; hence it could only be used to prove the existence of a good solution. This is very similar to the Santa Claus problem from the previous chapter. There, a quasi-polynomial variant by

Polácek et al. [62] and a polynomial variant by Annamalai et al. [7] were later discovered.

In this chapter, we start with a simple quasi-polynomial time $(2+\epsilon)$-approximation in order to introduce some ideas. We then present a simpler variant of Svensson's non-constructive algorithm, which in addition achieves a better approximation ratio; thereby we improve the bound on the integrality gap of the configuration LP. Finally, we combine both approaches in a very sophisticated $(11/6+\epsilon)$-approximation algorithm that terminates in quasi-polynomial time. This leads to the first better-than-2 approximation algorithm for Restricted Assignment, which does not need exponential running time. The algorithm is purely combinatorial and uses the configuration-LP only in the analysis. The last algorithm implies the results of the previous two. Nevertheless, they are significantly less complex and already demonstrate many of the key techniques used in the last algorithm.

COMPARISON TO RELATED ALGORITHMS. In Svensson's algorithm [68] and ours, jobs are moved until the desired allocation is found. The presentations of the algorithms differ significantly, but on a high level Svensson's algorithm is closely related to the second (exponential time) algorithm we give. Considering simplified instances with only jobs of two sizes, both algorithms and their analysis are essentially the same. Our exponential time algorithm is a cleaner adaption to general instances, which is much less technical and has an better approximation ratio.

The approach for obtaining a quasi-polynomial running time resembles that of [62], where it was done for the restricted Santa Claus problem. In the Restricted Assignment problem, however, this turns out to be significantly more challenging. The basic idea in both algorithms is to reduce the search depth to a logarithmic value. A fundamental structure in both cases are chains of big jobs or resources. Big jobs have a size greater than $1/2$ times the optimal makespan. Informally, a chain of big jobs is a sequence $j_1, i_1, j_2, i_2, j_3, i_3 \ldots$, where j_1 is a big job allowed to be placed on i_1; j_2 is a big job currently assigned to i_1, which is also allowed on i_2; j_3 is a big job currently assigned to i_2, but also allowed on i_3; etc. A similar situation can arise in the restricted Santa Claus problem, except that big resources (the counter-part to jobs) have a size at least the desired solution value. It turns out that these chains play a critical role and in the Santa Claus problem they have a very simple structure. This is because on a player (the counter-part to a machine) which has one big resource we do not place any other resources. In the Restricted Assignment case it is necessary to place also other jobs on machines that have big jobs. This means that the simple operation of moving every job of a chain to the next machine works well in the Santa Claus case, but in the Restricted Assignment case this can result in bad machines (machines that have too much load), if we are not careful. Perhaps this is also a reason why at this time there is no polynomial time better-than-2 approximation algorithm known for Restricted Assignment and the only

progress in this direction is in the case where we have only one big job size. Note that in this case, chains are again simple.

NOTATION. For a set of jobs $A \subseteq \mathcal{J}$, we write $p(A)$ in place of $\sum_{j \in A} p_j$. For other variables indexed by jobs, we may do the same. An allocation is a function $\sigma : \mathcal{J} \to \mathcal{M}$, where $\sigma(j) \in \Gamma(j)$ for all $j \in \mathcal{J}$. We write $\sigma^{-1}(i)$ for the set of all jobs j which have $\sigma(j) = i$.

CONFIGURATION LP. Similar to the SANTA CLAUS problem from the previous chapter, a configuration LP with an exponential size can be constructed, which can be solved approximately in polynomial time with a rate of $(1+\epsilon)$ for every $\epsilon > 0$ [12]. For every machine i and every $\tau \geq 0$ let

$$\mathcal{C}(i,\tau) = \{S \subseteq \mathcal{J} : p(S) \leq \tau \text{ and for all } j \in S, i \in \Gamma(j)\}.$$

These are the configurations for machine i and makespan τ. They are a set of jobs that have volume at most τ and can run on machine i. The optimum OPT^* of the configuration LP is the lowest τ such that the following linear program is feasible.

Primal of the configuration LP for RESTRICTED ASSIGNMENT

$$\begin{aligned} \sum_{C \in \mathcal{C}(i,\tau)} x_{i,C} &\leq 1 && \forall i \in \mathcal{M} \\ \sum_{i \in \mathcal{M}} \sum_{C \in \mathcal{C}(i,\tau) : j \in C} x_{i,C} &\geq 1 && \forall j \in \mathcal{J} \\ x_{i,C} &\geq 0 \end{aligned}$$

This linear program assigns at least one configuration to every machine and makes sure that every job is assigned at least once. We will also construct the dual after adding the objective $\max (0,\dots,0) \cdot x$ to the configuration LP:

Dual of the configuration LP for RESTRICTED ASSIGNMENT

$$\begin{aligned} \min \sum_{i \in \mathcal{M}} y_i - \sum_{j \in \mathcal{J}} z_j & \\ \sum_{j \in C} z_j \leq y_i & \qquad \forall i \in \mathcal{M}, C \in \mathcal{C}(i,\tau) \\ y_i, z_j \geq 0 & \end{aligned}$$

Recall, the value τ is a constant in the LP and, if the configuration LP is infeasible with τ, this means $\mathrm{OPT}^* > \tau$. Furthermore, we can derive the following condition from duality.

Lemma 38. *Let $y \in \mathbb{R}^{\mathcal{M}}_{\geq 0}$ and $z \in \mathbb{R}^{\mathcal{J}}_{\geq 0}$ such that $\sum_{i \in \mathcal{M}} y_i < \sum_{j \in \mathcal{J}} z_j$ and for every $i \in \mathcal{M}$ and $C \in \mathcal{C}(i,\tau)$ it holds that $\sum_{j \in C} z_j \leq y_i$, then $\mathrm{OPT}^* > \tau$.*

It is easy to see that if such a solution y, z exists, then every component can be scaled by a constant to obtain a feasible solution lower than any given

value. Hence, the dual must be unbounded and therefore the primal must be infeasible.

6.1 SIMPLE ALGORITHM

In this section we present a quasi-polynomial time $(2+\epsilon)$-approximation algorithm. It should be noted that there do exist clean and simple polynomial time 2-approximation algorithms for this problem and even for the general problem of scheduling on unrelated machines. The purpose of this section is merely to introduce some concepts and how they can be used to get a quasi-polynomial running time. We use a dual approximation framework where we perform a binary search over variable $\tau \in [p_{\max}, n \cdot p_{\max}]$, where $p_{\max} = \max_{j \in \mathcal{J}} p_j$. In each iteration we either prove that $\mathrm{OPT}^* > \tau$ or find an allocation with makespan at most $(1+\epsilon)\tau$. We stop the binary search once upper and lower bound differ by less than a factor of $(1+\epsilon)$. This gives a $(1+\epsilon)^2$-approximation in $\log_{1+\epsilon}(n) = O(1/\epsilon \cdot \log(n))$ iterations of the binary search. By scaling down ϵ we can get a $(1+\epsilon)$-approximation in the same asymptotic time. The algorithmic idea for the inner method is basically a breadth-first search:

Let $\sigma : \mathcal{J} \to \mathcal{M}$ be an arbitrary allocation. We use layers $L_0, \dots, L_\ell$, which are disjoint sets of machines. We write $L_{\leq k}$ for the union over all machines in $L_0, \dots, L_k$. Further, let $\tilde{\mathcal{J}}(L_{\leq k}, \sigma)$ denote the set of jobs j with $\sigma(j) \in L_{\leq k}$. We call a machine i *good*, if $p(\sigma^{-1}(i)) \leq 2+\epsilon$, and *bad*, otherwise. The algorithm (see Alg. 2) initializes L_0 as the bad machines. Then the subsequent layers L_{k+1} are created as the union over all $\Gamma(j) \setminus L_{\leq k}$ where $j \in \tilde{\mathcal{J}}(L_{\leq k}, \sigma)$. If there is a machine with a load at most $(1+\epsilon)\tau$ in some layer, we move a job from a lower layer to this layer and then start from the beginning.

RUNNING TIME. We consider two consecutive iterations right before some job is moved and the layers are deleted. Let σ and σ' be the allocations at the earlier and at the later iteration. Likewise, let $L_{\leq \ell}$ and $L'_{\leq \ell'}$ be the layers. We show that the vector

$$(b', |\tilde{\mathcal{J}}(L'_{\leq 0}, \sigma)|, \dots, |\tilde{\mathcal{J}}(L'_{\leq \ell'}, \sigma)|, -1)$$

is lexicographically smaller than

$$(b, |\tilde{\mathcal{J}}(L_{\leq 0}, \sigma)|, \dots, |\tilde{\mathcal{J}}(L_{\leq \ell}, \sigma)|, -1),$$

where b' and b are the number of bad machines for σ' and σ. Since the number of layers is at most $\log_{1+\epsilon}(|\mathcal{M}|) = O(1/\epsilon \cdot \log(|\mathcal{M}|))$ and each component is bounded by $|\mathcal{J}|$, the overall running time is $|\mathcal{J}|^{O(1/\epsilon \cdot \log(|\mathcal{M}|))}$. For the lexicographic decrease, notice that the algorithm moves a job j to a machine i only when $p(\sigma^{-1}(i)) \leq (1+\epsilon)\tau$. Hence, after adding j, the load on i is $p(\sigma^{-1}(i)) + p_j \leq (1+\epsilon)\tau + \tau \leq (2+\epsilon)\tau$. Thus, the algorithm never turns a good machine into a bad machine and $b' \leq b$. If $b' < b$ we are done. Otherwise, $b' = b$ and since no jobs were moved to or from $L_{\leq \ell - 1}$, $L'_{\leq k} = L_{\leq k}$ and $\tilde{\mathcal{J}}(L'_{\leq k}, \sigma') = \tilde{\mathcal{J}}(L_{\leq k}, \sigma)$ for all $k \leq \min\{\ell - 1, \ell'\}$. If $\ell' \leq \ell - 1$, then the

Input: Instance $(\mathcal{J}, \mathcal{M}, (\Gamma_j)_{j\in\mathcal{J}}, (p_j)_{j\in\mathcal{J}})$
Result: Schedule $\sigma : \mathcal{J} \to \mathcal{M}$ with makespan $\leq (2+\epsilon)\tau$
or "err", if $\tau < \mathrm{OPT}^*$.
let σ be an arbitrary allocation ;
$\ell \leftarrow 0$;
let L_0 be the set of bad machines ;
while $L_0 \neq \emptyset$ **do**
let $L_{\ell+1}$ be the union over all $\Gamma(j) \setminus L_{\leq\ell}$ for $j \in \tilde{\mathcal{J}}(L_{\leq i}, \sigma)$;
if *there is an* $i \in L_{\ell+1}$ *with* $p(\sigma^{-1}(i)) \leq (1+\epsilon)\tau$ **then**
find a job j with $\sigma(j) \in L_{\leq\ell}$ and $i \in \Gamma(j)$;
$\sigma(j) \leftarrow i$;
delete $L_0, \ldots, L_{\ell+1}$;
$\ell \leftarrow 0$;
let L_0 be the new set of bad machines ;
else
$\ell \leftarrow \ell + 1$;
if $\ell \geq \log_{1+\epsilon}(|\mathcal{M}|)$ **then**
return "err" ;

Algorithm 2: Quasi-polynomial $(2+\epsilon)$-approximation algorithm

$\ell' + 1$-th component will be -1 and therefore smaller than $|\tilde{\mathcal{J}}(L_{\leq\ell'+1}, \sigma)|$. Otherwise, the ℓ-th component will be smaller because we moved one job away from $L_{\leq\ell}$.

CORRECTNESS. We have to verify that if the algorithm returns "err", then $\mathrm{OPT}^* > \tau$. We will do so using Lemma 38. Assume that $\ell \geq \log_{1+\epsilon}(|\mathcal{M}|)$ and let σ and $L_{\leq\ell}$ be the current allocation and layer structure. For every $i \in L_k$ define

$$y_i = (1+\epsilon)^{1-k}.$$

Furthermore, define $y_i = (1+\epsilon)^{-\log_{1+\epsilon}(|\mathcal{M}|)}$, if $i \notin L_{\leq\ell}$. For jobs $j \in \mathcal{J}$ set

$$z_j = (1+\epsilon)^{-k} \cdot p_j / \tau,$$

where k is minimal with $j \in \tilde{\mathcal{J}}(L_{\leq k}, \sigma)$ and $z_j = 0$ if there is no such k. We need to show that $\sum_{j\in\mathcal{J}} z_j > \sum_{i\in\mathcal{M}} y_i$ and for all $i \in \mathcal{M}$ and $C \in \mathcal{C}(i,\tau)$ it holds that $z(C) \leq y_i$. For the former, we argue

$$\begin{aligned}
\sum_{j\in\mathcal{J}} z_j &= \sum_{i\in\mathcal{M}} z(\sigma^{-1}(i)) = \sum_{k=0}^{\ell} \sum_{i\in L_k} (1+\epsilon)^{-k} \frac{p(\sigma^{-1}(i))}{\tau} \\
&> (2+\epsilon)|L_0| + \sum_{k=1}^{\ell} (1+\epsilon)^{-k}(1+\epsilon)|L_k| \geq 1 + \sum_{k=0}^{\ell} (1+\epsilon)^{1-k}|L_k| \\
&= |\mathcal{M}| \cdot (1+\epsilon)^{-\log_{1+\epsilon}(|\mathcal{M}|)} + \sum_{k=0}^{\ell} (1+\epsilon)^{1-k}|L_k| \geq \sum_{i\in\mathcal{M}} y_i.
\end{aligned}$$

For the latter condition, first consider a machine $i \in L_k$ and $C \in \mathcal{C}(i,\tau)$. There can be no job $j \in C$ with $j \in \tilde{\mathcal{J}}(L_{\leq k-2},\sigma)$, since otherwise i would be in an earlier layer. Therefore, $z_j \leq (1+\epsilon)^{-(k-1)} p_j$ for all $j \in C$. Thus,

$$z(C) \leq (1+\epsilon)^{1-k} \frac{p(C)}{\tau} \leq (1+\epsilon)^{1-k} = y_i.$$

Now let $i \in \mathcal{M} \setminus L_{\leq \ell}$ and let $C \in \mathcal{C}(i,\tau)$. No job in C can be in $\tilde{\mathcal{J}}(L_{\leq \ell-1},\sigma)$. This means, $z_j \leq (1+\epsilon)^{-\ell} p_j$ for all $j \in C$ and

$$z(C) \leq (1+\epsilon)^{-\ell} \frac{p(C)}{\tau} \leq (1+\epsilon)^{-\log_{1+\epsilon}(|\mathcal{M}|)} = y_i.$$

Using the lemma this implies that $\mathrm{OPT}^* > \tau$.

6.2 NON-CONSTRUCTIVE INTEGRALITY GAP BOUND

In this section, we give an approximation algorithm with ratio $11/6$. The algorithm is similar to the previous one, but it adds only single moves, instead of a whole layers of reachable machines. This leads to an exponential running time bound. Hence, the algorithm in this section only gives a non-constructive bound on the integrality gap of the configuration LP.

6.2.1 *Algorithm*

Given an allocation $\sigma : \mathcal{J} \to \mathcal{M}$, we call a machine i *bad*, if $p(\sigma^{-1}(i)) > 11/6 \cdot \tau$. A machine is *good*, if it is not bad. We define *big* jobs to be those $j \in \mathcal{J}$ that have $p_j > 1/2 \cdot \tau$ and small jobs all others.

As the previous one, this algorithm starts with an arbitrary allocation and moves jobs until all machines are good, or it can prove that the configuration LP is infeasible w.r.t. τ. During this process, a machine that is already good will never be made bad.

The central data structure of the algorithm is an ordered list of pending moves $P = (P_1, P_2, \ldots, P_\ell)$. Here, every component $P_k = (j,i)$, $j \in \mathcal{J}$ and $i \in \Gamma(j)$, stands for a move the algorithm wants to perform. It will not perform the move, if this would create a bad machine, i.e., $p(\sigma^{-1}(i)) + p_j > 11/6 \cdot \tau$. If it does not create a bad machine, we say that the move (j,i) is valid. For every $0 \leq k \leq \ell$ define $L_{\leq k} := (L_1, \ldots, L_k)$, the first k elements of L (with $L_{\leq 0}$ being the empty list).

Depending on the current allocation σ and list of pending moves $P_{\leq \ell}$, we define a binary relation $R(P_{\leq \ell},\sigma) \subseteq \mathcal{J} \times \mathcal{M}$. For a pair $(j,i) \in R(P_{\leq \ell},\sigma)$ we say machine i *repels* j w.r.t. $P_{\leq \ell}$. This does not mention σ and therefore slightly abuses notation, but during the lifetime of $P_{\leq \ell}$ the allocation σ does not change and is always clear from the context.

The definition of repelled jobs is given later. The algorithm will only add a new move (j,i) to the current list P, if j is repelled by its current machine and not repelled by the target i w.r.t. P (see Algorithm 3).

In the algorithm we use a lexicographic order (p_j, j, i) of the moves (j, i). Here, we assume that there is an arbitrary order on jobs and machines, which is consistent throughout the iterations.

Input: Instance $(\mathcal{J}, \mathcal{M}, (\Gamma_j)_{j\in\mathcal{J}}, (p_j)_{j\in\mathcal{J}})$ of RESTRICTED ASSIGNMENT, $\tau \geq \mathrm{OPT}$
Result: Schedule $\sigma : \mathcal{J} \to \mathcal{M}$ with makespan $\leq 11/6 \cdot \tau$.
let σ be an arbitrary allocation ;
$\ell \leftarrow 0$; // number of pending moves
while *there is a bad machine* **do**
  choose a move $(j, i) \notin P_{\leq \ell}$, $j \in \mathcal{J}$ and $i \in \Gamma(j)$, where j is repelled by $\sigma(j)$ and not repelled by i w.r.t. $P_{\leq \ell}$ and (p_j, j, i) is lexicographically minimal among all candidates;
  $P_{\ell+1} \leftarrow (j, i)$;
  $\ell = \ell + 1$;
  if $p(\sigma^{-1}(i)) + p_j \leq 11/6 \cdot \tau$ **then**
    $\sigma(j) \leftarrow i$;
    delete $P_1, \ldots, P_\ell$;
    $\ell \leftarrow 0$;

Algorithm 3: Algorithm for RESTRICTED ASSIGNMENT

REPELLED JOBS. We define the repelled jobs of each machine inductively w.r.t. $P_{\leq k}$, $k = 0, 1, \ldots, \ell$.

(INITIALIZATION) If $k = 0$, let every bad machine i repel every job j w.r.t. $P_{\leq k}$.

(MONOTONICITY) If i repels j w.r.t. $L_{\leq k}$, then let i repel j also w.r.t. $P_{\leq k+1}$.

The remaining rules regard $k > 0$ and we let $(j_k, i_k) := P_k$, i.e., the last move added. In order to make space for j_k, the machine i_k should repel jobs.

(SMALL-ALL) If j_k is small, let i_k repel all jobs.

In the case that j_k is big, we need to be more careful. It helps to imagine that the algorithm is a lazy one: It repels jobs only if it is really necessary.

For $i \in \mathcal{M}$ let $S_i(P_{\leq k-1}, \sigma)$ be those small jobs j which have $\sigma(j) = i$ and which are repelled by all other potential machines, i.e., $\Gamma(j) \setminus \{i\}$, w.r.t. $P_{\leq k-1}$. The intuition behind $S_i(P_{\leq k-1}, \sigma)$ is that we do not expect that i can get rid of any of these jobs.

Next, define a threshold W_0 as the minimum $W \geq 0$ such that the small jobs in $S_{i_k}(P_{\leq k-1}, \sigma)$ and all big jobs below this threshold are already too large to move j_k, i.e.,

$$p(S_{i_k}(L_{\leq k-1}, \sigma)) + p(\{j \in \sigma^{-1}(i_k) : 1/2 < p_j \leq W\}) + p_{j_k} > 11/6 \cdot \tau.$$

Furthermore, define $W_0 = \infty$ if no such W exists. In order to make (j_k, i_k) valid, it is necessary (although not always sufficient) to remove one of the big jobs with size at most W_0. Hence, we define,

(BIG-ALL) if j_k is big and $W_0 = \infty$, then let i_k repel all jobs w.r.t $P_{\leq k}$ and

(BIG-BIG) if j_k is big and $W_0 < \infty$, then let i_k repel $S_{i_k}(L_{\leq k-1}, \sigma)$ and all jobs j with $1/2 < p_j \leq W_0$.

Note that repelling $S_{i_k}(P_{\leq k-1}, \sigma)$ seems unnecessary, since those jobs do not have any machine to go to. However, this definition simplifies the analysis. It is also notable that the special case where $W_0 = 0$ is equivalent to $p(S_{i_k}(L), \sigma) + p_{j_k} > 11/6 \cdot \tau$ and here the algorithm gives up making (j_k, i_k) valid. Finally, we want to highlight the following counter-intuitive (but intentional) aspect of the algorithm. It might happen that some job of size greater than W_0 is moved to i_k, but has to be moved again later in order to make (j_k, i_k) valid.

6.2.2 *Analysis*

Lemma 39. *If the configuration LP is feasible for τ and there is a bad machine, the algorithm always finds a move to execute.*

Proof. Suppose toward contradiction, there is a bad machine, no move in $P_{\leq \ell}$ is valid and no move can be added to $P_{\leq \ell}$. We will construct values $(z_j)_{j \in \mathcal{J}}$, $(y_i)_{i \in \mathcal{M}}$ with the properties as in Lemma 38 and thereby show that the configuration LP is infeasible. During this proof, σ and $P_{\leq \ell}$ are constant and refer to the state at which the algorithm is stuck. We omit $P_{\leq \ell}$ when we say i repels j.

Let $\tilde{\mathcal{J}}_i$ denote all jobs $j \in \sigma^{-1}(i)$ that are repelled by i. We write $\tilde{\mathcal{J}} = \bigcup_{i \in \mathcal{M}} \tilde{\mathcal{J}}_i$. For every $j \in \mathcal{J}$ let

$$z_j = \begin{cases} \min\left\{\frac{p_j}{\tau}, \frac{5}{6}\right\} & \text{if } j \in \tilde{\mathcal{J}} \text{ and} \\ z_j = 0 & \text{otherwise.} \end{cases}$$

Let $y_i := 1$ if $i \in \mathcal{M}$ repels all jobs and $y_i = z(\sigma^{-1}(i))$ otherwise.

Let $i \in \mathcal{M}$ and $C \in \mathcal{C}(i, \tau)$. We need to show that $z(C) \leq y_i$. If $y_i = 1$ this follows immediately, because $z(C) \leq p(C)/\tau \leq 1$. We assume w.l.o.g. that i does not repel all jobs and thus $y_i = z(\sigma^{-1}(i))$. In particular, i does not repel small jobs that are on other machines. This means that $z_j = 0$ for every small job $j \in C \setminus \sigma^{-1}(j)$. Otherwise, (j, i) could be added to $P_{\leq \ell}$. If there are no big jobs in C, we therefore get $z(C) \leq z(\sigma^{-1}(i)) = y_i$. Clearly, there can be at most one big job $j_B \in C$, since such a job has $p_{j_B} > 1/2 \cdot \tau$ and C cannot have a load greater than τ. If $z_{j_B} = 0$ or $\sigma(j_B) = i$ the argument above still holds.

We recap: The only interesting case is when $y_i = z(\sigma^{-1}(i))$, there is exactly one big job $j_B \in C \setminus \sigma^{-1}(i)$, and $z_{j_B} = \min\{p_{j_B}/\tau, 5/6\}$.

CASE 1: i REPELS j_B. Since i is not the target of a small job move and it is good (otherwise it would repel all jobs), there must be a big job move that causes i to repel j_B. In other words, there is a move $(j_k, i) = P_k$ such that

$j_B \leq W_0$, where W_0 is as in the definition of repelled jobs for $P_{\leq k}$. Recall that $W_0 < \infty$ is the minimal W with

$$p(S_i(P_{\leq k-1}, \sigma)) + p(\{j \in \sigma^{-1}(i) : 1/2 < p_j \leq W\}) + p_{j_k} > 11/6 \cdot \tau.$$

Since $W_0 \geq p_{j_B}$ and it is minimal, there must be a big job $j'_B \in \sigma^{-1}(i)$ with $p_{j'_B} = W_0 \geq p_{j_B}$ and j'_B is also repelled by i (because $p_{j'_B} \leq W_0$). We get

$$z(C) \leq z(C \setminus \{j_B\}) + z_{j_B} \leq z(\sigma^{-1}(i) \setminus \{j'_B\}) + z_{j'_B} = z(\sigma^{-1}(i)) = y_i.$$

CASE 2: i DOES NOT REPEL j_B. Since (j_B, i) cannot be added to P, it must already be in P. Let $P_k = (j_B, i)$ and W_0 as in the definition of repelled edges w.r.t. $P_{\leq k}$. Then

$$p(S_i(P_{\leq k-1}, \sigma)) + p(\{j \in \sigma^{-1}(i) : 1/2 < p_j \leq W_0\}) + p_{j_B} > 11/6 \cdot \tau.$$

If $W_0 \geq 5/6$, then there is some $j'_B \in \sigma^{-1}(i)$ with $p_{j'_B} = W_0 \geq 5/6$. Similar to the previous case, it follows that

$$z(C) \leq z(C \setminus \{j_B\}) + z_{j_B} \leq z(\sigma^{-1}(i) \setminus \{j'_B\}) + 5/6 = z(\sigma^{-1}(i)) = y_i.$$

If $W_0 \leq 5/6$, then all of the considered jobs have $z_j = p_j/\tau$, i.e.,

$$\begin{aligned} y_i &= z(\sigma^{-1}(i)) \\ &\geq z(S_i(P_{\leq k-1}, \sigma)) + z(\{j \in \sigma^{-1}(i) : 1/2 < p_j \leq W_0\}) \\ &= p(S_i(P_{\leq k-1}, \sigma))/\tau + p(\{j \in \sigma^{-1}(i) : 1/2 < p_j \leq W_0\})/\tau \\ &> 11/6 - p_{j_B}/\tau \geq 5/6 + (\tau - p_{j_B})/\tau \geq z(C). \end{aligned}$$

The last inequality holds, because the z_{j_B} is at most $5/6$ and the volume of the small jobs in C (in particular, their value) is at most $(\tau - p_{j_B})/\tau$. It remains to show that $\sum_{j \in \mathcal{J}} z_j > \sum_{i \in \mathcal{M}} y_i$. We prove that, with amortization, good machines satisfy $z(\sigma^{-1}(i)) \geq y_i$ and on bad machines strict inequality holds. Let i be a bad machine. Then i repels all jobs (in particular, those in $\sigma^{-1}(i)$). Hence,

$$z(\sigma^{-1}(i)) \geq 5/6 \cdot p(\sigma^{-1}(i))/\tau > 55/36 > 1 = y_i.$$

For good machines that do not repel all jobs, equality holds by definition. We will partition those good machines that do repel all jobs into those $i \in \mathcal{M}$ which have $(j_S, i) \in P_{\leq \ell}$ for a small job j_S and those that do not.

Fact 40. *At least half of the machines i that repel all jobs are target of a small job, i.e., $(j_S, i) \in P_{\leq \ell}$ for some small job j_S.*

We argue that whenever a move $(j_B, i) = P_k$ of a big job j_B is added, it is not valid, and $W_0 = \infty$ in the definition of repelled jobs w.r.t. $P_{\leq k}$, then there is some small job move (j_S, i') that can be added to P. Since the algorithm prefers small job moves over big ones, the next move after P_k will necessarily

be a small job move. Since no two small job moves can be added for the same target, the fact follows. If $W_0 = \infty$, this means that

$$p(S_i(L_{\leq k-1}, \sigma)) + p(\{j \in \sigma^{-1}(i) : p_j > 1/2\}) + p_{j_B} \leq 11/6 \cdot \tau.$$

Since the move (j_B, i) is not valid, however, we also have that

$$p(\sigma^{-1}(i)) + p_{j_B} > 11/6 \cdot \tau.$$

This implies that there must be a small job $j_S \in \sigma^{-1}(i) \setminus S_i(L_{\leq k-1}\sigma)$. In particular, there exists some $i' \in \Gamma(j_S) \setminus \{i\}$ by which j_S is not repelled. It is also not repelled by i' w.r.t. $P_{\leq k}$, since P_k only adds repelled jobs for i. Therefore, (j_S, i') is a candidate for the next move to be added after $P_{\leq k}$. This concludes the proof for the fact.

Let i be a machine that repels all jobs, but is not target of a small job. Then there is a big job j_B with $(j_B, i) \in P_{\leq \ell}$ and this move is not valid. Either there is a job $j \in \sigma^{-1}(i)$ with $z_j = 5/6$ or $z_j = p_j/\tau$ for all $j \in \sigma^{-1}(i)$. Thus,

$$\begin{aligned} z(\sigma^{-1}(i)) &\geq \min\left\{\frac{5}{6}, \frac{p(\sigma^{-1}(i))}{\tau}\right\} \\ &\geq \min\left\{\frac{5}{6}, \frac{11/6 - p_{j_B}}{\tau}\right\} \geq \frac{5}{6} = y_i - \frac{1}{6}. \end{aligned}$$

Next, let i be a machine such that there exists a small job j_S with $(j_S, i) \in P_{\leq \ell}$. This move is also not valid. In the following, we distinguish between the cases where $\sigma^{-1}(i)$ has no job j with $z_j = 5/6$, one such job, or at least two. Note that all jobs have $p_j \leq \tau$.

$$\begin{aligned} z(\sigma^{-1}(i)) &\geq \min\left\{\frac{p(\sigma^{-1}(i))}{\tau}, \frac{p(\sigma^{-1}(i)) - \tau}{\tau} + \frac{5}{6}, \frac{10}{6}\right\} \\ &\geq \min\left\{\frac{11}{6} - \frac{p_{j_S}}{\tau} - 1 + \frac{5}{6}, \frac{10}{6}\right\} \geq \frac{7}{6} = y_i + \frac{1}{6}. \end{aligned}$$

Because of Fact 40, we can amortize and get

$$\sum_{j \in \mathcal{J}} z_j = \sum_{i \in \mathcal{M}} z(\sigma^{-1}(i)) > \sum_{i \in \mathcal{M}} y_i. \qquad \square$$

Lemma 41. *The algorithm terminates.*

Proof. Consider two consecutive iterations of the main loop right before a move is executed and the list of moves P is deleted. Let σ, $P_{\leq \ell}$ be the allocation and list of pending moves in the former iteration and σ', $P'_{\leq \ell'}$ in the latter. Let b and b' be the number of bad machines in σ and σ'. Recall, $R(P_{\leq k}, \sigma) \subseteq \mathcal{J} \times \mathcal{M}$ is the set of all i, j where j is repelled by i w.r.t. $P_{\leq k}$. Further, let $\tilde{\mathcal{J}}(P_{\leq k}, \sigma)$ denote all jobs j that are repelled by $\sigma(j)$ w.r.t. $P_{\leq k}$. For each prefix of P (and of P') we define a potential function

$$\Phi(P_{\leq k}, \sigma) = (p_{j_k}, j_k, i_k, |\mathcal{J} \times \mathcal{M}| - |R(P_{\leq k}, \sigma)|, |\tilde{\mathcal{J}}(P_{\leq k}, \sigma)|)$$

We claim that the vector

$$(b', |\tilde{\mathcal{J}}(P'_{\leq 0}, \sigma')|, \Phi(P'_{\leq 0}, \sigma'), \ldots, \Phi(P'_{\leq \ell'}, \sigma'), -1)$$

is lexicographically smaller than

$$(b, |\tilde{\mathcal{J}}(P_{\leq 0}, \sigma)|, \Phi(P_{\leq 0}, \sigma), \ldots, \Phi(P_{\leq \ell}, \sigma), -1).$$

Note that the length of the vector is bounded by $5 \cdot |\mathcal{M}| \cdot |\mathcal{J}| + 3$, since no move appears twice in the list and every component can have at most $|\mathcal{J}| \cdot |\mathcal{M}| + 1$ different values. Thus, the number of possible vectors is finite and hence the algorithm terminates.

Recall that the algorithm never turns a good machine bad, which means $b' \leq b$. If $b' < b$, we are done. Likewise, if $b = b'$ and some job is moved from a bad machine to a good machine, then $|\tilde{\mathcal{J}}(P'_{\leq 0}, \sigma')| < |\tilde{\mathcal{J}}(P_{\leq 0}, \sigma)|$ and again the first vector is lexicographically smaller. We can therefore focus on the case $b' = b$, $\tilde{\mathcal{J}}(P'_{\leq 0}, \sigma') = \tilde{\mathcal{J}}(P_{\leq 0}, \sigma)$, and $\sigma(j) = \sigma'(j)$ for all $j \in \tilde{\mathcal{J}}(P_{\leq 0}, \sigma)$. The rest of the argument is by induction. Let $k \leq \min\{\ell - 1, \ell'\}$ and assume that

1. $P'_{\leq k-1} = P_{\leq k-1}$,
2. $R(P'_{\leq k-1}, \sigma') = R(P_{\leq k-1}, \sigma)$.
3. $\tilde{\mathcal{J}}(P'_{\leq k-1}, \sigma') = \tilde{\mathcal{J}}(P_{\leq k-1}, \sigma)$; $\sigma'(j) = \sigma(j)$ for all $j \in \tilde{\mathcal{J}}(P_{\leq k-1}, \sigma)$,

We will show that $\Phi(L'_{\leq k}) \leq \Phi(L_{\leq k})$ (lexicographically) and if equality holds, then (1), (2), and (3) also hold for k. This implies the lexicographical decrease: If $\ell' < \ell$ it follows easily. This is because the prefix of the first vector ending in $\Phi(P'_{\leq \ell'})$ is lexicographically not bigger than the prefix of the second vector ending in $\Phi(P_{\leq \ell'})$. Furthermore, the next component is -1 in the first vector, but something non-negative in the other. Now consider the case $\ell' \geq \ell$. Let $(j_\ell, i_\ell) = P_\ell$ be the move that was executed. Then $\sigma'(j_\ell) = i_\ell \neq \sigma(j_\ell)$. Furthermore, j_ℓ is repelled by $\sigma(j_\ell)$ w.r.t. $L_{\leq \ell-1}$. Hence, (2) cannot hold with $k - 1 = \ell - 1$ and thus we cannot have $\Phi(L'_{\leq k}) = \Phi(L_{\leq k})$ for all k.

The approach for the induction is to show that if $\Phi(L'_{\leq k}, \sigma')$ is not smaller than $\Phi(L_{\leq k}, \sigma)$, $(j_k, i_k) = P_k$ also has to be selected as P'_k. In that case, no job can have been moved to i_k, if it is repelled by i_k w.r.t. $L_{\leq k}$. From the way they are chosen, this implies the jobs which i_k repels as a consequence of P_k are also repelled in the rules for P'_k. If they do not increase, the number of jobs j with $\sigma(j)$ that are repelled by i_k cannot increase. Let us now formalize this argument.

Notice that $(j_\ell, i_\ell) = P_\ell$ is the move that was executed, i.e., $\sigma'(j_\ell) = i_\ell$ and by construction of $P_{\leq \ell}$, j_ℓ is not repelled by i_ℓ w.r.t. $P_{\leq \ell-1}$ (in particular, not w.r.t. $P_{\leq k}$ $(*)$). Let $(j_k, i_k) = P_k$. By (1) we have that $(j_k, i_k) \notin P_{\leq k-1} = P'_{\leq k-1}$. Since j_k is repelled by $\sigma(j_k)$ w.r.t. $L_{\leq k}$, by (2) we have that $\sigma'(j_k) = \sigma(j_k)$. By (3) it is not repelled by i_k w.r.t. $L'_{\leq k-1}$. Therefore, (j_k, i_k) is was a candidate for P'_k. Either this or a move (j, i), where (p_j, j, i) is lexicographically smaller than (p_{j_k}, j_k, i_k) is chosen. In the latter case we have $\Phi(L'_{\leq k}) < \Phi(L_{\leq k})$.

Hence, assume that $P'_k = (j_k, i_k)$. This means (1) holds for k. Note that since (2) and (3) hold for $k-1$, we only have to check the consequences of P'_k and P_k. In other words, we have to check whether the rules for move P'_k imply the same repelled jobs as P_k and whether some job repelled due to this rule has been moved.

If j_k is small, then i_k repels all jobs w.r.t. $L'_{\leq k}$ and $L_{\leq k}$ and therefore (3) also holds for k. Job j_ℓ cannot have been moved to i_k (see $(*)$). If it was moved away from i_k, then $\tilde{J}(P'_{\leq k}, \sigma') \subsetneq \tilde{J}(P_{\leq k-1}, \sigma)$. Otherwise, equality holds and no job was moved in this set, i.e., (2) holds for k.

Now assume j_k is big. First, we argue that $S'_{i_k}(L'_{\leq k-1}, \sigma') = S_{i_k}(L_{\leq k-1}, \sigma)$. Recall, $S_{i_k}(L_{\leq k-1}, \sigma)$ are the small jobs j with $\sigma(j) = i_k$ and j is repelled by all machines in $\Gamma(j) \setminus \{i_k\}$ w.r.t. $L_{\leq k}$. Let $j \in S'_{i_k}(L'_{\leq k-1}, \sigma')$. Assume toward contradiction that $\sigma(j) \neq i_k = \sigma'(j)$. Then $j = j_\ell$ and $i_k = i_\ell$. However, $\sigma(j_\ell)$ does not repel j_ℓ w.r.t. $P_{\leq k-1}$. Otherwise (j_ℓ, i_ℓ) would have been chosen instead of (j_k, i_k) as P_k. By (3) $\sigma(j_\ell)$ also does not repel j_ℓ w.r.t. $P'_{\leq k-1}$. Hence, $j \notin S'_{i_k}(L'_{\leq k-1}, \sigma')$, a contradiction. Consequently, $\sigma(j) = i_k$. By (3) it follows that $j \in S_{i_k}(L_{\leq k-1}, \sigma)$. Let $j \in S_{i_k}(L_{\leq k-1}, \sigma)$. Since it is repelled by all potential machines w.r.t. $P_{\leq k-1}$, there cannot be a move for j in a later layer. In particular j_ℓ cannot be j. This means $\sigma'(j) = \sigma(j) = i_k$ and by (3) $j \in S_{i_k}(L_{\leq k-1})$. We conclude that $S_{i_k}(L_{\leq k-1}) = S'_{i_k}(L'_{\leq k-1})$. Let W_0 be the minimal the minimal $W \geq 0$ with

$$\underbrace{p(S_{i_k}(L_{\leq k-1}, \sigma))}_{=p(S'_{i_k}(L'_{\leq k-1}, \sigma'))} + p(\{j \in \sigma^{-1}(i) : 1/2 < p_j \leq W\}) + p_{j_k} > 11/6 \cdot \tau,$$

or ∞ if no such W exists. Since all jobs j with $1/2 < p_j \leq W_0$ are repelled by i_k w.r.t. $P_{\leq k}$, it follows by $(*)$ that

$$p(\{j \in \sigma'^{-1}(i) : 1/2 < p_j \leq W\}) \leq p(\{j \in \sigma^{-1}(i) : 1/2 < p_j \leq W\})$$

It follows that W'_0, the minimal $W \geq 0$ with

$$p(S'_{i_k}(L'_{\leq k-1}, \sigma')) + p(\{j \in \sigma'^{-1}(i) : 1/2 < p_j \leq W\}) + p_{j_k} > 11/6 \cdot \tau,$$

is at least as big as W_0. In particular, if $W_0 = \infty$, then i_k repels all jobs w.r.t. $P_{\leq k}$ and w.r.t. $P'_{\leq k}$. Otherwise, by definition i_k repels all jobs in $S_i(L_{\leq k-1}, \sigma)$ and all jobs j with $1/2 < p_j \leq W_0$ w.r.t. $L_{\leq k}$. By $W'_0 \geq W_0$ these jobs are also repelled by i_k w.r.t. $L'_{\leq k}$. Hence $R(P'_{\leq k}, \sigma') \supseteq R(P_{\leq k}, \sigma)$. If equality holds then because of $(*)$, $\tilde{J}(P'_{\leq k}, \sigma') \supseteq \tilde{J}(P_{\leq k}, \sigma)$. If equality also holds here, then none of these jobs are moved and (1), (2), and (3) hold. If equality does not hold at some point, $\Phi(L'_{\leq k-1}, \sigma') < \Phi(L_{\leq k-1}, \sigma)$. □

Theorem 42. *The integrality gap of the configuration LP for* Restricted Assignment *is at most* $11/6$.

6.3 QUASI-POLYNOMIAL TIME ALGORITHM

The previous running time bound is clearly exponential. In this section, we will improve the running time to $n^{O(1/\epsilon \cdot \log(n))}$ for an approximation rate of

$11/6 + 2\epsilon$, where $n = |\mathcal{M}| + |\mathcal{J}|$ and $\epsilon > 0$ can be chosen arbitrarily. Note that by scaling ϵ, the coefficient of 2 can be removed.

6.3.1 *Algorithm*

Our approach for turning the running time quasi-polynomial is to combine the two algorithms presented in the previous sections. Instead of adding only one candidate move at a time like in the exponential time algorithm, we add them all. The set of these moves is called a layer. After a layer is added, re-evaluate the repelled jobs and again construct a layer of all sensible moves. This simple approach described has some major issues that we have to handle carefully using more sophisticated techniques as described in the following. First, however, we make some technical preparations. We will call a machine i *bad*, if $p(\sigma^{-1}(i)) > (11/6 + 2\epsilon)\tau$ and *good*, otherwise. As before, we call jobs j small, if $p_j \leq 1/2 \cdot \tau$ and big otherwise. Further, we distinguish big jobs into medium, which have $p_j \leq 5/6 \cdot \tau$, and huge, which have $p_j > 5/6 \cdot \tau$. Unlike in the previous algorithms we establish the invariant that at all times the current allocation assigns at most one huge job to each machine. An initial allocation that satisfies this can easily be found via bipartite matching with the huge jobs on one side and the machines on the other. It is maintained by the following definition of a valid huge job move.

Definition 8 (Valid huge move)**.** *Let $j \in \mathcal{J}, i \in \Gamma(j) \setminus \{\sigma(j)\}$ with $p_j > 5/6 \cdot \tau$. (j,i) is a valid move, if $p(\sigma^{-1}(i)) + p_j \leq (11/6 + 2\epsilon)\tau$ and there is no huge job in $\sigma^{-1}(i)$.*

For the remaining jobs we define the moves as follows.

Definition 9 (Valid non-huge move)**.** *Let $j \in \mathcal{J}, i \in \Gamma(j) \setminus \{\sigma(j)\}$ with $p_j \leq 5/6 \cdot \tau$. (j,i) is a valid move, if*

1. *$p(\sigma^{-1}(i)) + p_j \leq (11/6 + 2\epsilon)\tau$ and $\sigma^{-1}(i)$ contains no huge job, or*
2. *$p(\{j' \in \sigma^{-1}(i) : p_j \leq 5/6 \cdot \tau\}) + p_j \leq (5/6 + 2\epsilon)\tau$ and $\sigma^{-1}(i)$ contains one huge job.*

Each valid move (j,i) satisfies $p(\sigma^{-1}(i)) + p_j \leq (11/6 + 2\epsilon)\tau$, i.e., each good machine stays good. (2) needs further elaboration. One could falsely assume that this establishes an invariant which says the non-huge load is at most $(5/6 + 2\epsilon) \cdot \tau$ on a machine with a huge job. This would be a marvelous invariant, if it could be guaranteed. However, a valid huge move can break this property. Therefore, (2) only gives something weaker. It's purpose is that when a machine has a huge job and a low non-huge load, then it will stay this way for as long as the huge job remains on the machine. This is to keep edges in the leap graph intact, a technique that will be elaborated later.

LAYERS. We will operate in layers $L_1, \ldots, L_\ell$ like in the first algorithm. These, however, will not only contain machines, but also fine grained moves like in the second algorithm. Again, we define a binary relation $R(L_{\leq k}, \sigma) \subseteq$

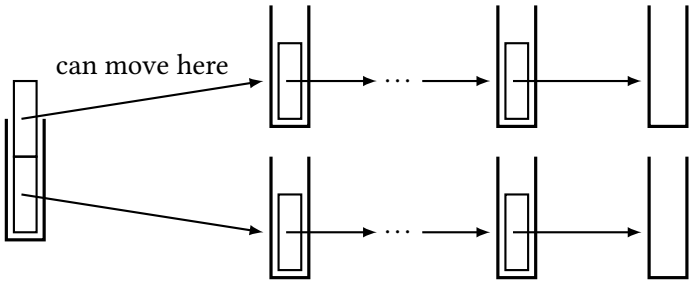


Figure 5: Example for linear number of layers

$\mathcal{J} \times \mathcal{M}$, which states that i repels j w.r.t. $L_{\leq k}$, if $(j,i) \in R(L_{\leq k}, \sigma)$. In the previous algorithms, the local search is mostly stateless, i.e., it searches for an improvement of σ without remembering anything from the past. Here we make a small exception. We maintain an order π on $\mathcal{J} \times \mathcal{M}$. When the algorithm adds certain critical moves, they are moved to the front of π. This will hint at the algorithm to do the same in the next iteration. It helps to argue about the running time, since the layers created this way are more consistent throughout the iterations. Finally, layers come in different forms. They can be leap layers, critical layers, small layers, or non-critical layers. This will become more clear in the actual description of the algorithm.

LEAPS. A straight-forward example shows that if we are using only simple moves like in the previous two algorithms, the number of layers needs to grow linearly. This would be a problem for obtaining a quasi-polynomial running time.

In the example (see Figure 5) the leftmost machine has two jobs j_1, j_1' of size τ assigned to it, which make the machine bad. The jobs each have a chain of machines connected to it: j_1 can go to a machine $i_1 \in \Gamma(j_1)$. On i_1 there is another job j_2 of size τ which can go to a machine $i_2 \in \Gamma(j_2)$, etc. At some point this chain ends with an empty machine. The same construction is made for j_1'. In order to make the bad machine good, either the top chain of jobs or the bottom chain has to be traversed. Hence, it seems like the number of layers would be roughly half the jobs or machines. It turns out that this problem only occurs with huge jobs and we will carefully circumvent it.

The leap technique is intended for moving such a chain of huge jobs at once. For an easy description we construct a directed bipartite graph $G(\sigma) = (V, E(\sigma))$ where $V = B \cup \mathcal{M}$, i.e., the vertices are big jobs B and the machines. There is an edge $(j_B, i) \in E(\sigma)$ if $i \in \Gamma(j_B) \setminus \{\sigma(j_B)\}$ and

$$p(\{j \in \sigma^{-1}(i) : p_j \leq 5/6\}) + p_{j_B} \leq (11/6 + 2\epsilon)\tau,$$

i.e., if there is no huge job in $\sigma^{-1}(i)$ the move (j_B, i) is valid. Furthermore, we let $(i, j_H) \in E(\sigma)$, if j_H is huge and $i = \sigma(j_H)$.

Suppose that there is some path $j_1, i_1, j_2, i_2, \ldots, j_k, i_k$ in G, where no huge job is assigned to i_k. Then we can move j_1 to i_1, j_2 from i_1 to i_2, j_3 from i_2 to i_3, etc. We will call this a *leap*.

The general theme for using this graph is the following. A big job j_B is repelled by $\sigma(j_B)$. Then all machines that are reachable by some path from j_B should repel their huge jobs as well. When one of them is removed, we can instantly free i from j_B. This way we are not going to put all moves of the path sequentially into the layers and avoid making it unnecessarily long.

Definition 10 (Valid leap)**.** *Let $j_1, i_1, \ldots, j_r, i_r$ be a path in the leap graph. It is called valid leap, if (j_r, i_r) is a valid move.*

By definition of the leap graph and the fact that every machine has at most one huge job, for a valid leap the following moves are all valid if executed in reverse order, i.e., $(j_r, i_r), (j_{r-1}, i_{r-1}), \ldots, (j_1, i_1)$.

Finally, we define a graph $G(L_{\leq k}, \sigma)$ which has all edges from $G(\sigma)$ of the form (i, j_H), but only the edges (j_B, i) where i does not repel j_B w.r.t. $L_{\leq k}$. The definition of repelled edges will be given later.

DESCRIPTION OF THE ALGORITHM. We are now ready to state the algorithm (see Alg. 4). It starts with an allocation σ with the property that each machine has at most one huge job assigned to it. The allocation of huge jobs can be found using bipartite matching and the remaining jobs are assigned arbitrarily. We initialize π as an arbitrary permutation of $\mathcal{J} \times \mathcal{M}$. Until all machines are good the algorithm searches for valid moves or leaps that improve σ. For this purpose we build layers. The layers are alternating between leap layers, critical layers, small layers, and non-critical layers: Layer L_{4k+1} is always a leap layer; L_{4k+2} is a critical layer; L_{4k+3} is a small layer; L_{4k+3} is a non-critical layer. A leap-layer $L_{\ell+1}$ consists of the machines that are reachable in the leap graph $G(L_{\leq \ell}, \sigma)$ by a job that is repelled by its current machine w.r.t. $L_{\leq \ell}$. For the critical layer $L_{\ell+2}$ and non-critical layer $L_{\ell+4}$, we select all (j_B, i) where j_B is a big job repelled by $\sigma(j_B)$, but not by i w.r.t. $L_{\leq \ell+1}$. A subset of these is taken in $L_{\ell+2}$. We will define below precisely how they are chosen, but they depend on π giving priority to the moves in the front. After they are selected, the critical moves are pushed to the front of π. All moves (j_S, i), where j_S is repelled by $\sigma(j_S)$, but not by i w.r.t. $L_{\leq \ell+2}$ are put in $L_{\ell+3}$. Finally, the previously considered big job moves (j_B, i) which were not taken in $L_{\ell+2}$ and where i still does not repel j_B (now w.r.t. $L_{\leq \ell+3}$) are taken in the non-critical layer $L_{\ell+4}$. If at any point a valid leap or move is found, it is executed and the structure of layers is reset. Note that the meaning of the *continue* statement in the pseudo-code is to jump to the next iteration of the while loop.

REPELLED JOBS. We define the repelled jobs of each machine inductively w.r.t. $L_{\leq k}$, $k = 0, 1, \ldots, \ell$.

(INITIALIZATION) Let the bad machine repel every job w.r.t. $L_{\leq 0}$.

(MONOTONICITY) If i repels j w.r.t. $L_{\leq k}$, then let i repel j also w.r.t. $L_{\leq k+1}$.

The remaining rules regard $k > 0$ and we define repelled jobs for a layer L_k. A layer L_k may be a leap layer, a critical layer, a small layer, or a non-critical

Input: Instance $(\mathcal{J}, \mathcal{M}, (\Gamma_j)_{j\in\mathcal{J}}, (p_j)_{j\in\mathcal{J}})$
Result: Schedule $\sigma : \mathcal{J} \to \mathcal{M}$ with makespan $\leq (11/6 + 2\epsilon)\tau$ or "err", if $\tau < \mathrm{OPT}^*$.
let σ be an allocation with at most one huge job on each machine ;
let π be an arbitrary order on $\mathcal{J} \times \mathcal{M}$;
$\ell \leftarrow 0$;
while *there is a bad machine* **do**
  if $\ell \geq 4\lceil \log_{1+\epsilon}(4|\mathcal{M}|)\rceil = O(1/\epsilon \cdot \log(|\mathcal{M}|))$ **then**
    return "err" ;
  let L^L_{new} be the set of machines reachable in the leap graph $G(L_{\leq\ell}, \sigma)$ by a big job j repelled by $\sigma(j)$ w.r.t. $L_{\leq\ell}$;
  $L_{\ell+1} \leftarrow (L^L_{\text{new}}, \textsc{leap})$;
  if *there is a machine i in $L_{\ell+1}$ with no huge job in $\sigma^{-1}(i)$* **then**
    let $j_1, i_1, \ldots, j_r, i_r = i$ be a path in $G(L_{\leq\ell}, \sigma)$, where j_1 is repelled by $\sigma^{-1}(j_r)$ w.r.t. $L_{\leq\ell}$;
    $\sigma(j_r) \leftarrow i_r, \ldots, \sigma(j_1) \leftarrow i_1$;
    delete $L_1, L_2, \ldots, L_{\ell+1}$; $\ell \leftarrow 0$; **continue**;
  let L^B_{new} be the set of all (j_B, i), $j_B \in \mathcal{J}$ big and $i \in \Gamma(j_B)$, with j_B repelled by $\sigma(j_B)$ and not by i w.r.t. $L_{\leq\ell+1}$;
  $L^C_{\text{new}} \leftarrow \texttt{CriticalMoves}(L^B_{\text{new}}, \sigma, L_{\leq\ell+1}, \pi)$;
  $L_{\ell+2} \leftarrow (L^C_{\text{new}}, \textsc{critical})$;
  move L^C_{new} to the front of π (keeping their pairwise order) ;
  if *there exists a valid move (j, i) in $L_{\ell+2}$* **then**
    $\sigma(j) \leftarrow i$;
    delete $L_1, L_2, \ldots, L_{\ell+2}$; $\ell \leftarrow 0$; **continue**;
  let L^S_{new} be the set of all (j_S, i), $j_S \in \mathcal{J}$ small and $i \in \Gamma(j_S)$, with j_S repelled by $\sigma(j_S)$ and not by i w.r.t. $L_{\leq\ell+2}$;
  $L_{\ell+3} \leftarrow (L^S_{\text{new}}, \textsc{small})$;
  if *there exists a valid move (j, i) in $L_{\ell+3}$* **then**
    $\sigma(j) \leftarrow i$;
    delete $L_1, L_2, \ldots, L_{\ell+3}$; $\ell \leftarrow 0$; **continue**;
  let L^{NC}_{new} be the set of all $(j, i) \in L^B_{\text{new}} \setminus L^C_{\text{new}}$ where i does not repel j w.r.t. $L_{\leq\ell+3}$;
  $L_{\ell+4} \leftarrow (L^{NC}_{\text{new}}, \textsc{non-critical})$;
  if *there exists a valid move (j, i) in $L_{\ell+4}$* **then**
    $\sigma(j) \leftarrow i$;
    delete $L_1, L_2, \ldots, L_{\ell+4}$; $\ell \leftarrow 0$; **continue**;
  $\ell \leftarrow \ell + 4$;
return σ ;

Algorithm 4: Constructive algorithm for Restricted Assignment

layer. In the first case, L_k contains a set of machines reachable in the leap graph. We define:

(LEAP) If L_k is a leap layer, let every machine i in L_k repel all big jobs that are adjacent to i in the leap graph $G(\sigma)$—that is, all huge jobs in $\sigma^{-1}(i)$ and all big jobs j_B with $i \in \Gamma(j_B)$ and

$$p(\{j' \in \sigma^{-1}(i) : p_{j'} \leq 5/6\}) + p_{j_B} \leq (11/6 + \epsilon)\tau.$$

(CRITICAL) If L_k is a critical layer, for every move (j,i) in L_k let i repel all jobs.

(SMALL) If L_k is a small layer, for every move (j,i) in L_k let i repel all jobs.

Now assume that L_k is a non-critical-layer and consider a move (j,i) in L_k. In the non-critical case the algorithm is lazy: It repels jobs only if it is really necessary. We first identify a set of small jobs that is unlikely to be moved. For $i \in \mathcal{M}$ define $S_i(L_{\leq k-1}, \sigma)$ to be the small jobs $j \in \sigma^{-1}(i)$ which are repelled by all machines in $\Gamma(j) \setminus \{i\}$.

Next, define a threshold W_0 as the minimum $W \geq 0$ such that the small jobs in $S_i(L_{\leq k-1}, \sigma)$ and all big jobs below this threshold are already too large to add j, i.e.,

$$p\left(\left\{j' \in \sigma^{-1}(i) : \frac{1}{2} < p_{j'} \leq W\right\}\right) + p(S_i(L_{\leq k-1}, \sigma)) + p_j > \left(\frac{11}{6} + 2\epsilon\right)\tau.$$

It will follow from the selection of critical moves that such a W_0 always exists and, moreover, $W_0 \leq 5/6$. When none of the jobs in $S_i(L_{\leq k-1}, \sigma)$ can be removed, it is necessary (although not always sufficient) to remove one of the big jobs with size at most W_0 in order to make (j,i) valid. Hence, we define,

(NON-CRITICAL) if L_k is a non-critical layer then for every move (j,i) let i repel all jobs j with $1/2 < p_j \leq W_0$ (where W_0 is defined as above) and all jobs in $S_i(L_{\leq k-1}, \sigma)$.

It is notable that the corner case where $W_0 = 0$ is equivalent to

$$p(S_i(L_{\leq k-1}, \sigma)) + p_j > (11/6 + 2\epsilon)\tau$$

and here the algorithm gives up making (j,i) valid. In particular, no additional big jobs will be repelled.

Finally, we want to highlight the following counter-intuitive (but intentional) aspect of the algorithm. It might happen that some job of size greater than W_0 is moved to i, only to be removed again in a later iteration, when W_0 has increased.

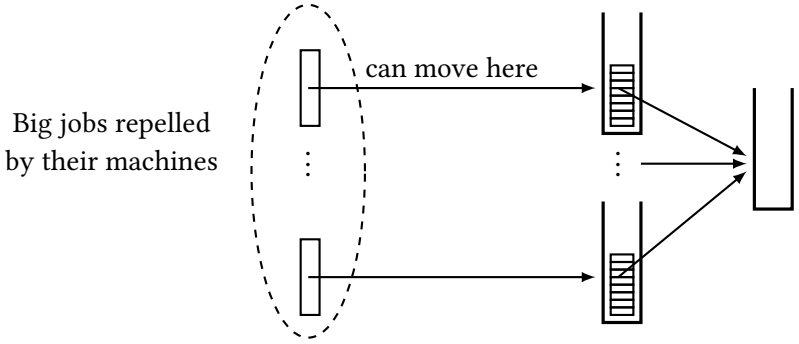


Figure 6: Bottleneck for small jobs

CRITICAL MOVE SELECTION. Suppose we are given some layers $L_{\leq \ell+1}$ and big job moves $(j,i) \in L^B_{\text{new}}$ where j is repelled by $\sigma(j)$ w.r.t. $L_{\leq \ell+1}$, but not by i. Which of these moves should be critical? Recall that for critical moves (j,i) the target machine i always repels all jobs. As in the exponential time algorithm, we later need to amortize these moves with small job moves. Hence, we should select critical moves in a way that they produce many small job moves.

In the following let $\overline{\mathcal{M}}$ denote the set of machines that repel all jobs w.r.t. $L_{\leq \ell+1}$. As a prime example of a situation we want to avoid, consider the following: There are a lot of critical moves (j,i) where $p_j = 1$, but on i there is a load of small jobs with volume slightly above $(11/6+2\epsilon)\tau - p_j = (5/6+2\epsilon)\tau \ll \tau$. Moreover, these small jobs j_S cannot go anywhere (meaning later layers will not have moves for them), because all their potential machines are in $\overline{\mathcal{M}}$, i.e., $\Gamma(j_S) \setminus \{i\} \subseteq \overline{\mathcal{M}}$. Hence, there will not be any machines to amortize this low load. We should consider small jobs like this as blocked volume and when there is too much blocked volume on i, a move (j,i) should not be critical. However, it is not enough to consider small jobs that have nowhere to go. It might also be that a lot of small jobs have only very few machines to go to. In the example above, imagine that all the small jobs share only one machine $i' \notin \overline{\mathcal{M}}$ to which they could go (see Figure 6). Then the average load is still very low. This is also something we want to avoid.

So how do we avoid these situations? We select the critical edges sequentially (see Algorithm 5). For the already added critical moves (j,i), we consider all machines reachable by a small job on i can go to as blocked as well, i.e., we add them to $\overline{\mathcal{M}}$. We add (j,i) to the critical moves only when the blocked small jobs (as described above) and the medium jobs on i have a volume that allows j to be added to i, if there were no other jobs.

6.3.2 *Analysis*

Lemma 43. *If the configuration LP is feasible for τ and there remains a bad machine, then within the first $\ell \leq 4\lceil \log_{1+\epsilon}(4|\mathcal{M}|)\rceil$ layers there will be a valid leap or move.*

Input: Moves L^B_{new}, schedule σ, Layers $L_{\leq \ell+1}$, and order π
Result: Critical moves $C \subseteq L^B_{\text{new}}$
$C \leftarrow \emptyset$;
let $\overline{\mathcal{M}}$ be the set of the machines that repel all jobs w.r.t. $L_{\leq \ell+1}$;
for $(j,i) \in L^B_{\text{new}}$ *ordered by* π **do**
  Small $\leftarrow p(\{j' \in \sigma^{-1}(i) : p_{j'} \leq 1/2 \text{ and } \Gamma(j') \setminus \{i\} \subseteq \overline{\mathcal{M}}\})$;
  Med $\leftarrow p(\{j' \in \sigma^{-1}(i) : 1/2 < p_{j'} \leq 5/6\})$;
  if $i \notin \overline{\mathcal{M}}$ *and* Small + Med + $p_j \leq 11/6 + 3\epsilon$ **then**
    $C \leftarrow C \cup \{(j,i)\}$;
    $\overline{\mathcal{M}} \leftarrow \overline{\mathcal{M}} \cup \{i\}$;
    for $j' \in \sigma^{-1}(i)$ *small* **do**
      $\overline{\mathcal{M}} \leftarrow \overline{\mathcal{M}} \cup \Gamma(j')$;
return C ;

Algorithm 5: Selection of critical moves

Proof. Suppose toward contradiction, there are bad machines, no move in $L_{\leq \ell}$ is valid, and $\ell = 4\lceil \log_{1+\epsilon}(4|\mathcal{M}|)\rceil$. We will construct values $(z_j)_{j\in\mathcal{J}}$, $(y_i)_{i\in\mathcal{M}}$ with the properties as in Lemma 38 and thereby show that the configuration LP is infeasible. Throughout the proof the allocation σ refers to the allocation in the iteration where no move or leap is found.

First, we define values $z_j^{(k)}, y_i^{(k)}$ for all prefixes of the layers ending in a leap layer, i.e., for each $L_{\leq 4k+1}$ with $0 \leq k < \ell/4$. Furthermore, for technical reasons we define the values $z_j^{(-1)}, y_i^{(-1)}$ as well as $y_i^{(\ell/4)}$. Then z_j, y_i will be set as a positive linear combination of these values.

Let $\tilde{\mathcal{J}}(L_{\leq 4k+1})$ denote all jobs j that are repelled by $\sigma(j)$ w.r.t. $L_{\leq 4k+1}$. For every $0 \leq k < \ell/4$ and $j \in \mathcal{J}$ let

$$z_j^{(k)} = \begin{cases} \min\left\{\frac{p_j}{\tau}, \frac{5}{6}\right\} & \text{if } j \in \tilde{\mathcal{J}}(L_{\leq 4k+1}), \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, let $y_i^{(k)} := 1+\epsilon$, if i repels all jobs w.r.t. $L_{\leq 4k+1}$ and $y_i^{(k)} := z^{(k)}(\sigma^{-1}(i))$, otherwise. Finally, define the corner cases $y_i^{(-1)} = 0$, $z_j^{(-1)} = 0$, and $y_i^{(\ell/4)} := 1+\epsilon$ for all i, j.

Notice that $z_j^{(-1)} \leq z_j^{(0)} \leq \cdots \leq z_j^{(\ell/4)}$ for all j (and the same holds for all $y_i^{(k)}$). We set

$$z_j^{(\leq k)} = \sum_{k'=-1}^{k} \frac{1}{(1+\epsilon)^{k'}} \cdot z_j^{(k')},$$

$$y_i^{(\leq k)} = \sum_{k'=-1}^{k} \frac{1}{(1+\epsilon)^{-k'}} \cdot y_i^{(k')}.$$

The coefficients decrease exponentially with the layer number. As we will see, this makes to the last values negligibly small (as in the first algorithm). Finally, set $z_j = z_j^{(\leq \ell/4-1)}$ and $y_i = y_i^{(\leq \ell/4)}$.

Claim 44. *Let $-1 \le k < \ell/4$, $i \in \mathcal{M}$ and $C \in \mathcal{C}(i,\tau)$. Then*

$$z^{(\le k)}(C) \le y_i^{(\le k+1)}.$$

In particular, this implies $z(C) = z^{(\le \ell/4-1)} \le y_i^{(\le \ell/4)} = y_i$ for all i, C.

Claim 45.

$$\sum_{j\in\mathcal{J}} z_j > \sum_{i\in\mathcal{M}} y_i.$$

Together the claims imply $\tau < \mathrm{OPT}^*$, a contradiction. □

Before we prove the claims, we will state the following auxiliary lemmata.

Lemma 46. *In an iteration where no valid move or leap is found consider the set L^B_{new} selected in the algorithm after a leap layer $L_{\ell+1}$ is created and let $(j_B, i) \in L^B_{\text{new}}$. Then*

$$p\left(\left\{j \in \sigma^{-1}(i) : p_j \le \frac{5}{6}\right\}\right) + p_{j_B} > \left(\frac{11}{6} + 2\epsilon\right)\tau.$$

Proof. Suppose toward contradiction that this does not hold. Then (j_B, i) is in the leap graph $G(\sigma)$. It is also in $G(L_{\le\ell}, \sigma)$, since i does not repel j_B w.r.t. $L_{\le\ell+1}$. Otherwise, (j_B, i) would not be in L^B_{new}. Obviously i is reachable by j_B in $G(L_{\le\ell}, \sigma)$. We argue that i is reachable by some big job repelled by its current machine w.r.t. $L_{\le\ell}$. This implies that i repels j_B w.r.t. $L_{\le\ell+1}$ by definition of repelled edges for a leap layer. This is a contradiction, since (j_B, i) could not be in L^B_{new} then. We know that $\sigma(j_B)$ repels j_B w.r.t. $L_{\le\ell+1}$. If it repels j_B already w.r.t. $L_{\le\ell}$, this follows trivially. Otherwise, j_B is repelled by i because of the (leap) rule in the definition of repelled edges for $L_{\ell+1}$. This can only be when $i \in L_{\ell+1}$, which means it is reachable by some big job repelled by its machine w.r.t. $L_{\le\ell}$. □

Lemma 47. *In an iteration where no valid move or leap is found consider the set L^B_{new} selected in the algorithm after a leap layer $L_{\ell+1}$ is created and let $(j_B, i) \in L^B_{\text{new}}$. Then*

$$p\left(\left\{j \in \sigma^{-1}(i) : \frac{1}{2} < p_j \le \frac{5}{6}\right\}\right) + p(S_i(L_{\le\ell+3}, \sigma)) + p_{j_B} > \left(\frac{11}{6} + 2\epsilon\right)\tau,$$

This lemma implies that the threshold W_0 chosen in the definition of repelled edges always exists and $W_0 \le 5/6$.

Proof. When (j_B, i) is a critical moves in $L_{\ell+2}$ or when $(j_S, i) \in L_{\ell+3}$ for some small job j_S, this is follows easily from the previous lemma, since $S_i(L_{\le\ell+3}, \sigma)$ contains all small jobs in $\sigma^{-1}(i)$. Each other move (j_B, i) would have been selected as a critical move, if this inequality did not hold: Consider the set $\overline{\mathcal{M}}$ in the selection of critical moves at the time (j_B, i) is considered. The algorithm adds (j_B, i) to the critical moves, if $i \notin \overline{\mathcal{M}}$ and

$$p\left(\left\{j \in \sigma^{-1}(i) : \frac{1}{2} < p_j \le \frac{5}{6}\right\}\right) + p(S'_i(\overline{\mathcal{M}}, \sigma)) + p_{j_B} \le \left(\frac{11}{6} + 2\epsilon\right)\tau, \quad (12)$$

where $S_i'(\overline{\mathcal{M}},\sigma)$ is the set of small jobs $j_S \in \sigma^{-1}(i)$ with $\Gamma(j_S)\setminus\{i\} \subseteq \overline{\mathcal{M}}$. Recall that all machines in $\overline{\mathcal{M}}$ either repel all jobs w.r.t. $L_{\leq \ell+1}$ or they are reachable by a small job on a machine that is target of a critical move. The latter kind must repel all jobs w.r.t. $L_{\leq \ell+3}$, because it is target of a small job move in $L_{\ell+3}$. Thus, $S_i'(\overline{\mathcal{M}},\sigma) \subseteq S_i(L_{\leq \ell+3},\sigma)$ and (12) is satisfied. Furthermore, $i \notin \overline{\mathcal{M}}$, since i does not repel all jobs w.r.t. $L_{\leq \ell+1}$ and we assumed that there is no small job j_S with $(j_S,i) \in L_{\ell+3}$. Thus, (j_B,i) would have been selected as a critical move. □

Proof of Claim 44. We argue inductively. The basis of the induction is trivial, since $z^{(\leq -1)}(C) = 0 \leq y_i^{(\leq 0)}$. Suppose that $k \geq 0$ and for all $k' < k$,

$$z^{(\leq k')}(C) \leq y_i^{(\leq k'+1)}$$

If $y_i^{(k+1)} \geq 1+\epsilon$ then immediately

$$\begin{aligned} z^{(\leq k)}(C) = z^{(\leq k-1)}(C) + (1+\epsilon)^{-k} z^{(k)}(C) &\leq y_i^{(\leq k)} + (1+\epsilon)^{-k}\frac{p(C)}{\tau} \\ &\leq y_i^{(\leq k)} + (1+\epsilon)^{-(k+1)} y_i^{(k+1)} = y_i^{(\leq k+1)}. \end{aligned}$$

We can therefore assume w.l.o.g. that $y_i^{(k+1)} = z^{(k+1)}(\sigma^{-1}(i))$. Thus, $k < \ell/4$ and i does not repel all jobs w.r.t. $L_{\leq 4(k+1)+1}$. Since by definition of repelled jobs, a machine that repels any small job from another machine always repels all jobs, we know that i does not repel small jobs that are on other machines w.r.t. $L_{\leq 4(k+1)+1}$. Hence, for all small jobs $j_S \in C \setminus \sigma^{-1}(i)$ it holds that $z_{j_S}^{(k)} = 0$: If this was not true, $\sigma(j_S)$ would repel j_S w.r.t. L_{4k+1}, in which case (j_S,i) would have been added to L_{4k+3} and i would repel all jobs, which is not true.

Consider the cases of big jobs in C. If there is none, then obviously every jobs in $C\setminus\sigma^{-1}(i)$ is small. Let $k' \leq k$. Then for all $j \in C\setminus\sigma^{-1}(i)$ it holds that $z_j^{(k')} \leq z_j^{(k)} = 0$. Consequently, $z^{(k')}(C) \leq z^{(k')}(\sigma^{-1}(i)) = y_i^{(k')}$. Hence,

$$z^{(\leq k)}(C) \leq z^{(\leq k)}(\sigma^{-1}(i)) = y_i^{(\leq k)} \leq y^{(\leq k+1)}{}_i.$$

Clearly, there can be at most one big job $j_B \in C$, since such a job has $p_{j_B} > 1/2\cdot\tau$ and C cannot have a volume greater than τ. If $z_{j_B}^{(k)} = 0$ or $j_B \in \sigma^{-1}(i)$, the argument above still works.

We recap: The crucial case is when $y_i^{(k+1)} = z^{(k+1)}(\sigma^{-1}(i))$, there is exactly one big job $j_B \in C\setminus\sigma^{-1}(i)$, and $z_{j_B}^{(k)} = \min\{p_{j_B}/\tau, 5/6\}$. Let $k' \leq k$ be minimal with $z_{j_B}^{(k')} = \min\{p_{j_B}/\tau, 5/6\}$. In particular, $z_{j_B}^{(-1)} = z_{j_B}^{(0)} = \cdots = z_{j_B}^{(k'-1)} = 0$.

CASE 1: i REPELS j_B W.R.T. $L_{\leq 4k'+1}$. This can either be because of a leap layer or a move layer in $L_{\leq 4k'+1}$. In the former case, there has to be a huge

job in $j_H \in \sigma^{-1}(i)$ which i repels w.r.t. $L_{\leq 4k'+1}$. Otherwise, there would be a valid path. Thus, for all $k'' \geq k'$ it holds that $z_{j_H}^{(k'')} = 5/6 \geq z_{j_B}^{(k'')}$ and

$$z^{(k'')}(C) = z_{j_B}^{(k'')} + z^{(k'')}(C \setminus \{j_B\}) \leq z_{j_H}^{(k'')} + z^{(k'')}(\sigma^{-1}(i) \setminus \{j_H\}) \leq z^{(k'')}(\sigma^{-1}(i)) \leq y_i^{(k'')}.$$

Furthermore, for all $k'' < k'$,

$$z^{(k'')}(C) = z^{(k'')}(C \setminus \{j_B\}) \leq z^{(k'')}(\sigma^{-1}(i)) \leq y_i^{(k'')}.$$

Hence, $z^{(\leq k)}(C) \leq y_i^{(\leq k)} \leq y_i^{(\leq k+1)}$.

Now consider the case in which there is some move $(j_{k'}, i)$ which causes i to repel j_B is w.r.t. $L_{\leq 4k'+1}$. The move $(j_{k'}, i)$ must be in a non-critical layer $L_{4k''+4}$, where $k'' < k'$, since i does not repel all jobs w.r.t. $L_{\leq 4k'+1}$. Let W_0 as in the definition of repelled jobs in consequence of $(j_{k'}, i)$ and let

$$R = \{j \in \sigma^{-1}(i) : 1/2 < p_j \leq W_0\} \cup S_i(L_{\leq 4(k''-1)+3}, \sigma),$$

The edges repelled by i because of $(j_{k'}, i)$ are exactly $S_i(L_{\leq 4k''+3}, \sigma)$ and all those j with $1/2 < p_j \leq W_0$. Hence, $p_{j_B} \leq W_0$. Recall that W_0 is chosen minimal with $p(R) + p_{j_{k'}} > (11/6 + 2\epsilon)\tau$. There must be a job $j_B' \in \sigma^{-1}(i)$ with $p_{j_B'} = W_0$, since otherwise W_0 would not be minimal. Thus, for all k''' it holds that $z_{j_B'}^{(k''')} \geq z_{j_B}^{(k''')}$ and

$$z^{(k''')}(C) = z_{j_B}^{(k''')} + z^{(k''')}(C \setminus \{j_B\}) \leq z_{j_B'}^{(k''')} + z^{(k''')}(\sigma^{-1}(i) \setminus \{j_B'\}) \leq y_i^{(k''')}.$$

It follows conveniently that $z^{(\leq k)}(C) \leq y_i^{(\leq k)} \leq y_i^{(\leq k+1)}$.

CASE 2: i DOES NOT REPEL j_B W.R.T. $L_{\leq 4k'+1}$. Since $z_{j_B}^{(k')} > 0$, j_B is repelled by $\sigma(j_B)$ w.r.t. $L_{\leq 4k'+1}$. Machine i does not repel all jobs w.r.t. $L_{\leq 4k'+1}$, which implies there is no move with target i in $L_{4k'+2}$ or $L_{4k'+3}$. Hence, (j_B, i) must be a move in layer $L_{4k'+4}$. Let

$$R = \{j \in \sigma^{-1}(i) : 1/2 < p_j \leq W_0\} \cup S_i(L_{\leq 4k'+1}, \sigma),$$

where W_0 is as in the definition of repelled jobs in consequence of (j_B, i). Then

$$p(R) + p_{j_B} > \left(\frac{11}{6} + 2\epsilon\right)\tau \geq p(C) + \left(\frac{5}{6} + 2\epsilon\right)\tau.$$

Furthermore, all jobs in R are repelled by i w.r.t. $L_{\leq 4k'+4}$ and therefore in $\tilde{\mathcal{J}}(L_{\leq 4(k'+1)+1})$. Since for all $j' \in R$ it holds that $p_{j'} \leq W_0 \leq 5/6$ (see Lemma 47), it follows that $z_{j'}^{(k''+1)} = p_{j'}/\tau$ for all $k'' \geq k'$. Thus,

$$\begin{aligned} z^{(k'')}(C) &= z_{j_B}^{(k'')} + z^{(k'')}(C \setminus \{j_B\}) \\ &\leq z_{j_B}^{(k'')} + (p(C) - p_{j_B})/\tau \\ &< z_{j_B}^{(k'')} + (p(R) - (5/6 + 2\epsilon)\tau)/\tau \\ &= z_{j_B}^{(k'')} + p(R)/\tau - 5/6 - 2\epsilon \\ &\leq (1-\epsilon)p(R)/\tau \\ &\leq (1-\epsilon)z^{(k''+1)}(\sigma^{-1}(i)) \leq \frac{z^{(k''+1)}(\sigma^{-1}(i))}{1+\epsilon}. \end{aligned}$$

Here we use that i is a good machine and therefore $p(R) \leq p(\sigma^{-1}(i)) \leq (11/6 + 2\epsilon)\tau < 2\tau$. We conclude,

$$\begin{aligned} z^{(\leq k)}(C) &= z^{(\leq k'-1)}(C) + \sum_{k''=k'}^{k} (1+\epsilon)^{-k''} z^{(k'')}(C) \\ &\leq y_i^{(\leq k')} + \sum_{k''=k'}^{k} (1+\epsilon)^{-k''} \frac{z^{(k''+1)}(\sigma^{-1}(i))}{1+\epsilon} \\ &\leq y_i^{(\leq k')} + \sum_{k''=k'}^{k} (1+\epsilon)^{-(k''+1)} y_i^{(k''+1)} = y_i^{(\leq k+1)}. \end{aligned}$$

□

Proof of Claim 45. Let i be a bad machine. Then i repels all jobs (in particular those in $\sigma^{-1}(i)$) w.r.t. $L_{\leq 0}$. Hence, for every $0 \leq k < \ell/4$ and $j \in \sigma^{-1}(i)$, $z_j^{(k)} = \min\{5/6, p_j/\tau\} \geq 5/6 \cdot p_j/\tau$ Thus,

$$z^{(k)}(\sigma^{-1}(i)) \geq \frac{5}{6} p(\sigma^{-1}(i))/\tau > \frac{5}{6}\left(\frac{11}{6} + 2\epsilon\right) > \frac{55}{36} + \epsilon > 1 + \epsilon + \frac{1}{2}.$$

This implies

$$\begin{aligned} y_i &= \sum_{k=0}^{\ell/4} (1+\epsilon)^{-k} y_i^{(k)} = \sum_{k=0}^{\ell/4-1} [(1+\epsilon)^{-k}(1+\epsilon)] + (1+\epsilon)^{-(\ell/4-1)} \\ &< \sum_{k=0}^{\ell/4-1} [(1+\epsilon)^{-k} \cdot z^{(k)}(\sigma^{-1}(i))] + (1+\epsilon)^{-(\ell/4-1)} - \frac{1}{2} \sum_{k=0}^{\ell/4-1} (1+\epsilon)^{-k} \\ &\leq z(\sigma^{-1}(i)) - \frac{1}{2}. \end{aligned}$$

In the last inequation, we use the last two elements of the sum $\sum_{k=0}^{\ell/4-1}(1+\epsilon)^{-k}$ to compensate for $(1+\epsilon)^{-(\ell/4-1)}$. The inequation shows that y_i is much smaller than $z(\sigma^{-1}(i))$. If for all good machines i and layers k we had $y_i^{(k)} = z^{(k)}(\sigma^{-1}(i))$ (which is the case when i does not repel all jobs w.r.t.

$L_{\leq k}$), the proof would be easy: $\sum_{j\in\mathcal{J}} z_j^{(\leq \ell/4-1)} = \sum_{i\in\mathcal{M}} z^{(\leq \ell/4-1)}(\sigma^{-1}(i))$ would be larger than $1/2 + \sum_{i\in\mathcal{M}} y_i^{(\leq \ell/4-1)}$. The former is exactly $\sum_{j\in\mathcal{J}} z_j$ and the latter is

$$1/2 + \sum_{i\in\mathcal{M}} y_i - \sum_{i\in\mathcal{M}} (1+\epsilon)^{-\ell/4} y_i^{(\ell/4)} < \sum_{i\in\mathcal{M}} y_i.$$

Here we use that the decrease in the coefficient makes $y_i^{(\ell/4)}$ neglectable, which we will explain in detail as we go through the actual proof.

Of course, there can be machines that repel all jobs and are set to $y_i^{(k)} = 1+\epsilon$. We have to make sure that they do not have a negative effect. Let B_k be the machines i with (j_B, i) in the k-th critical layer for some j_B, i.e., in L_{4k+1}. Let A_k be the machines i with (j_S, i) in the k-th small layer for some j_S, i.e., in L_{4k+3}.

Let $k < \ell/4$ and $i \in B_k$. i repels all jobs w.r.t. $L_{\leq 4(k+1)+1}$. Thus, $\sigma^{-1}(i) \subseteq \tilde{\mathcal{J}}(L_{\leq 4(k+1)+1})$. Let (j_B, i) as above. This move is not valid. Either there is a job $j \in \sigma^{-1}(i)$ with $z_j^{(k+1)} = 5/6$ or $z_j^{(k+1)} = p_j/\tau$ for all $j \in \sigma^{-1}(i)$. Thus,

$$\begin{aligned} z^{(k+1)}(\sigma^{-1}(i)) &\geq \min\{5/6,\ p(\sigma^{-1}(i))/\tau\} \\ &\geq \min\{5/6,\ 11/6 + 2\epsilon - p_{j_B}/\tau\} \geq 5/6 \geq 1 + \epsilon - \epsilon - 1/6 \\ &= y_i^{(k+1)} - \epsilon - 1/6. \end{aligned}$$

Next, let $i \in A_k$. Then there is a move $(j_S, i) \in L_{\leq 4k+1}$ with j_S small. Of course, this move is not valid either. In the following, we distinguish between the cases where $\sigma^{-1}(i)$ has no huge job or one huge job.

$$\begin{aligned} z^{(k+1)}(\sigma^{-1}(i)) &\geq \min\{p(\sigma^{-1}(i))/\tau,\ (p(\sigma^{-1}(i)) - \tau)/\tau + 5/6\} \\ &\geq 8/6 + 2\epsilon - 1 + 5/6 = \frac{7}{6} + 2\epsilon \geq y_i^{(k+1)} + 1/6 + 2\epsilon. \end{aligned}$$

The bounds above show that machines in A_k have $z^{(k+1)}(\sigma^{-1}(i))$ above $y_i^{(k+1)}$ and machines in B_k below. In order to amortize the machines, we have to proof a bounded ratio between them: We argue that for every $k < \ell/4$, $|A_k| \geq |B_k|$. Notice that $B_k \dot\cup A_k$ are exactly the machines that are added to $\overline{\mathcal{M}}$ in the selection of critical moves for L_{4k+2}. Hence, it suffices to show that at most half of them are target of critical big job moves. Consider a critical move (j_B, i) for a big job j_B that is added in the critical move selection. By Lemma 46

$$p\left(\left\{j \in \sigma^{-1}(i) : p_j \leq \frac{5}{6}\right\}\right) + p_{j_B} > \left(\frac{11}{6} + 2\epsilon\right)\tau.$$

Because the move (j_B, i) is selected as a critical move, it holds that

$$p\left(\left\{j \in \sigma^{-1}(i) : \frac{1}{2} < p_j \leq \frac{5}{6}\right\}\right) + p(S_i'(\overline{\mathcal{M}}, \sigma)) + p_{j_B} \leq \left(\frac{11}{6} + 2\epsilon\right)\tau,$$

where $S_i'(\overline{\mathcal{M}}, \sigma)$ are the small jobs $j_S \in \sigma^{-1}(i)$ with $\Gamma(j_S) \setminus \{i\} \subseteq \overline{\mathcal{M}}$ with $\overline{\mathcal{M}}$ as at the time before (j_B, i) is selected. Consequently, there is a small job $j_S \in$

$\sigma^{-1}(i) \setminus S'_i(\overline{\mathcal{M}}, \sigma)$. It follows that there exists a machine $i' \in \Gamma(j_S) \setminus (\overline{\mathcal{M}} \cup \{i\})$. The algorithm adds i and i' to $\overline{\mathcal{M}}$. In other words, whenever the algorithm adds a machine i to B_k, it adds at least one machine i' to A_k. It follows that

$$\begin{aligned}
\sum_{j \in \mathcal{J}} z_j &= \sum_{k=0}^{\ell/4-1} \sum_{i \in \mathcal{M}} (1+\epsilon)^{-k} z^{(k)}(\sigma^{-1}(i)) \\
&> \sum_{k=0}^{\ell/4-1} (1+\epsilon)^{-k} \left[\left(\frac{1}{6} + \epsilon \right) \underbrace{(|A_k| - |B_k|)}_{\geq 0} + \sum_{i \in \mathcal{M}} y_i^{(k)} \right] + \frac{1}{2} \\
&\geq \sum_{i \in \mathcal{M}} y_i + \underbrace{\frac{1}{2} - \sum_{i \in \mathcal{M}} (1+\epsilon)^{-\ell/4} y_i^{(\ell/4)}}_{\geq 0} \\
&\geq \sum_{i \in \mathcal{M}} y_i
\end{aligned}$$

In the last inequality we use that by choice of ℓ, $(1+\epsilon)^{\ell/4} \leq 4|\mathcal{M}|$, which implies

$$\sum_{i \in \mathcal{M}} (1+\epsilon)^{-\ell/4} y_i^{(\ell/4)} \leq 2|\mathcal{M}|(1+\epsilon)^{-\ell/4} \leq \frac{1}{2}. \qquad \square$$

Lemma 48. *The algorithm terminates in time $n^{O(1/\epsilon \log(n))}$, where $n = |\mathcal{J}| + |\mathcal{M}|$.*

Proof. We are looking at the states of two consecutive iterations right before a move or leap is performed. Let σ be the schedule in the former iteration and σ' in the latter. Likewise, define layers $L_{\leq \ell}$ and $L'_{\leq \ell'}$ right before they collapse. Let $\tilde{\mathcal{J}}(L_{\leq k}, \sigma)$ be the jobs j repelled by $\sigma(j)$ w.r.t. $L_{\leq k}$. Recall, $R(L_{\leq k}, \sigma)$ is the set of all $(j,i) \in \mathcal{J} \times \mathcal{M}$ where i repels j w.r.t. $L_{\leq k}$.

We define a potential function Φ for each of the layers. Set

$$\Phi(L_{\leq k}, \sigma) = \begin{cases} |R(L_{\leq k}, \sigma)| & \text{if } L_k \text{ is a leap layer,} \\ (|L_k|, |\mathcal{J}| - |\tilde{\mathcal{J}}(L_{\leq k}, \sigma)|) & \text{if } L_k \text{ is a critical layer,} \\ |\mathcal{J}| - |\tilde{\mathcal{J}}(L_{\leq k}, \sigma)| & \text{if } L_k \text{ is a small layer,} \\ (|R(L_{\leq k}, \sigma)|, |\mathcal{J}| - |\tilde{\mathcal{J}}(L_{\leq k}, \sigma)|) & \text{if } L_k \text{ is a non-critical layer.} \end{cases}$$

Claim 49. *The vector*

$$(g', |\mathcal{J}| - |\tilde{\mathcal{J}}(L'_{\leq 0}, \sigma')|, \Phi(L'_{\leq 1}, \sigma'), \ldots, \Phi(L'_{\leq \ell'}, \sigma'), \infty)$$

is lexicographically bigger than

$$(g, |\mathcal{J}| - |\tilde{\mathcal{J}}(L_{\leq 0}, \sigma)|, \Phi(L_{\leq 1}, \sigma), \ldots, \Phi(L_{\leq \ell}, \sigma), \infty)$$

Since the number of layers is at most $O(1/\epsilon \log(n))$ and components can have only $O(n^3)$ different values, the number of vectors is is bounded by $n^{O(1/\epsilon \log(n))}$. Since for every move or leap it decreases lexicographically, the lemma follows easily from the claim. $\square$

Proof of Claim 49. If the number of good machines increases, the claim follows immediately. If it does not, but a job is moved from a bad machine to a good one, then $\tilde{\mathcal{J}}'(L'_{\leq 0}, \sigma') \subsetneq \tilde{\mathcal{J}}(L_{\leq 0}, \sigma)$, i.e., the claim follows again. Hence, assume neither case is true. Let $1 \leq k \leq \min\{\ell - 1, \ell'\}$ and:

1. $L_{\leq k-1} = L'_{\leq k-1}$;
2. $\tilde{\mathcal{J}}(L_{\leq k-1}) = \tilde{\mathcal{J}}'(L'_{\leq k-1})$ and $\sigma(j) = \sigma'(j)$ for all $j \in \tilde{\mathcal{J}}(L_{\leq k-1})$;
3. $R(L'_{\leq k-1}, \sigma') = R(L_{\leq k}, \sigma)$.

We will prove: $\Phi(L'_{\leq k}) \geq \Phi(L_{\leq k})$ and if equality holds, (1), (2) and (3) also hold for k. This implies the claim by induction: If $\ell' < \ell$, then the prefix of the first vector ending in $\Phi(L'_{\leq \ell'}, \sigma)$ is lexicographically not smaller than the prefix of the second one ending in $\Phi(L_{\leq \ell'}, \sigma)$. Furthermore, the next component in the first vector is ∞, whereas it is something finite in the second. If $\ell' \geq \ell$, then we notice that (2) cannot hold for $k - 1 = \ell - 1$. This is because some leap or move in L_ℓ was executed and therefore a job j that is repelled by $\sigma(j)$ w.r.t. $L_{\leq \ell - 1}$ was moved.

CASE 1: L_k IS A LEAP LAYER. We argue that every machine i reachable in the leap graph $G(L_{\leq k-1}, \sigma)$ by a job big job j_0 repelled by $\sigma(j_0)$ w.r.t. $L_{\leq k-1}$ is also reachable by j_0 in the leap graph $G(L'_{\leq k-1}, \sigma')$. Because of (2), this means that the set of reachable machines in σ by any such job, which is exactly L_k, is a subset of L'_k. It suffices to show that every edge reachable by j_0 in $G(L_{\leq k-1}, \sigma)$ is also in the leap graph $G(L'_{\leq k-1}, \sigma')$. Because of (3) it suffices to show that it is in $G(\sigma')$. An edge in the leap graph can be one of two kinds. It can be from a machine to a huge job, i.e., (i, j_H), which exists because $\sigma(j_H) = i$. We argue that j_H was not not moved, which means $\sigma'(j_H) = \sigma(j_H) = i$ and therefore the edge is also in $G(\sigma')$. Suppose toward contradiction that a move (j_H, i') was executed. Then there is no huge job in $\sigma^{-1}(i')$ and $p(\sigma^{-1}(i')) + p_{j_H} \leq (11/6 + 2\epsilon) \cdot \tau$. Therefore i' would also be reachable by j_0 in $G(\sigma)$. Hence, the algorithm would execute a leap in L_k, which it did not, since $\ell > k$. Similarly, if j_H was moved as part of a leap, then all machines in this leap would be reachable already in L_k and there would have been a valid leap already.

Now consider an edge of the form (j_B, i). If i had no huge job in $\sigma^{-1}(i)$, then again, there would have been a valid leap in L_k, which cannot be. The edge (j_B, i) exists in the leap graph of σ because $i \in \Gamma(j_B) \setminus \{\sigma(j_B)\}$ and

$$p(\{j \in \sigma^{-1}(i) : p_j \leq 5/6\}) + p_{j_B} \leq (11/6 + 2\epsilon)\tau.$$

It could only be removed, if j_B was moved to i—this cannot be the case for the same reason as above—or some job j' with $p_{j'} \leq 5/6$ is moved to i. By definition of a valid non-huge move, however, this means that

$$\begin{aligned}(11/6 + 2\epsilon)\tau &\geq p(\{j \in \sigma^{-1}(i) : p_j \leq 5/6\}) + p_{j'} + 1 \\ &\geq p(\{j \in \underbrace{\sigma^{-1}(i) \cup \{p_{j'}\}}_{=\sigma'^{-1}(i)} : p_j \leq 5/6\}) + p_{j_B}.\end{aligned}$$

Therefore, (j_B, i) is also in $G(\sigma')$. We conclude, all reachable machines in L_k are also in L'_k, i.e., $L'_k \supseteq L_k$. By the arguments above, every job adjacent to a machine in L_k in $G(L_{\leq k-1}, \sigma)$ is also adjacent to this machine in L'_k in $G(L'_{\leq k-1}, \sigma')$. Thus, $R(L'_{\leq k}, \sigma') \supseteq R(L_{\leq k}, \sigma)$. If equality does not hold, then $\Phi(L'_{\leq k}, \sigma') > \Phi(L_{\leq k}, \sigma)$. Otherwise, (3) holds for k. (1) must also hold for k, because $L'_k \supsetneq L_k$ was true, then the additional machine would repel at least one additional job (its huge job). Finally, (2) holds, because no huge job on a reachable machine was moved as elaborated above.

CASE 2: L_k IS A CRITICAL LAYER. We show that every critical move in L_k is also in L'_k. By induction hypothesis, we know that the moves (j, i), $i \in \Gamma(j)$, where j is a big job repelled by $\sigma(j) = \sigma'(j)$, but not by i, w.r.t. $L_{\leq k-1}$ and w.r.t. $L'_{\leq k-1}$ are the same. Therefore, the sets L^B_{new} from which the critical moves are selected are the same in both cases. Recall that critical moves are added greedily in the order of π' (π). In π' the moves L^B_{new} are ordered in a way that first the moves from L_k appear (in the order of π) and then all others. This is because in the main algorithm when L_k was created, all $(j, i) \in L_k$ were moved to the front of π. We just have to understand that none of $L^B_{\text{new}} \setminus L_k$ were moved to the front at a later time. This is because there is no way that a move, which is not selected as critical, can be selected in a later layer.

Let $(j_1, i_1), \ldots, (j_{r-1}, i_{r-1})$ be the first $r-1$ critical moves selected in L_k. Furthermore, let $\overline{\mathcal{M}}_{r-1}$ and $\overline{\mathcal{M}}'_{r-1}$ be as in the algorithm before the r-th critical move was added. Note that before the first critical move was added, by (3) it holds that $\overline{\mathcal{M}}'_0 = \overline{\mathcal{M}}_0$, since these are the machines that repel all jobs w.r.t. $L_{\leq k-1}$. We assume for induction that $\overline{\mathcal{M}}'_{r-1} \subseteq \overline{\mathcal{M}}_{r-1}$ and that $(j_1, i_1), \ldots, (j_{r-1}, i_{r-1})$ were also added to L'_k. Because no move or leap in $L_{\leq k-1}$ was executed and i_r repels all jobs w.r.t. $L_{\leq k}$, we know that $\sigma'^{-1}(i_r) \subseteq \sigma^{-1}(i_r)$. In particular, every medium job in $\sigma'^{-1}(i_r)$ was already in $\sigma^{-1}(i_r)$. Moreover, every small job $j_S \in \sigma'^{-1}(i_r)$ with $\Gamma(j_S) \setminus \{i_r\} \subseteq \overline{\mathcal{M}}'_{r-1} \subseteq \overline{\mathcal{M}}_{r-1}$ was also in $\sigma^{-1}(i_r)$. Hence, the condition for adding (j_r, i_r) to L'_k holds, since it did for L_k. Finally, $\overline{\mathcal{M}}'_r$ is the union of $\overline{\mathcal{M}}'_{r-1}$, $\{i\}$, and $\Gamma(j_S)$ for every small $j_S \in \sigma'^{-1}(i_r)$. This is a subset of $\overline{\mathcal{M}}_{r-1}$, $\{i\}$, and $\Gamma(j_S)$ for every small $j_S \in \sigma^{-1}(i_r) \subseteq \sigma'^{-1}(i_r)$, which is $\overline{\mathcal{M}}_r$.

If $L'_k \supsetneq L_k$, nothing has to be shown, since $\Phi(L'_{\leq k}) > \Phi(L_{\leq k})$. Otherwise, $L'_k = L_k$ and therefore (3) follows for k directly. If some job was moved away from a machine of a critical move, then again $\Phi(L'_{\leq k}) > \Phi(L_{\leq k})$. Otherwise, (2) follows for k.

CASE 3: L_k IS A SMALL LAYER. As in the previous case, we have that $L_k = L'_k$, i.e., (1) holds also for k. (3) also holds for k, since in the rules of a small layer, every target of a move repels every job. This is the same in $L'_{\leq k}$ and $L_{\leq k}$. If some job was moved away from a target machine of a move in L_k, then $\tilde{\mathcal{J}}(L'_{\leq k}) \subsetneq \tilde{\mathcal{J}}(L_{\leq k})$ and therefore $\Phi(L'_{\leq k}) > \Phi(L_{\leq k})$. Otherwise, (2) follows for k as well.

CASE 4: L_k IS A NON-CRITICAL-LAYER. By the arguments in Case 2 we know that the previous critical moves and the moves they are chosen from are the same and therefore also for the non-critical moves $L'_k = L_k$. Let $(j,i) \in L_k$. We argue that

$$S_i(L_{\leq k-1}, \sigma) = S_i(L'_{\leq k-1}, \sigma').$$

Let $j_S \in S_i(L_{\leq k-1}, \sigma)$. Then there cannot be a move (j_S, i') in some higher layer than L_{k-1}. This is because j_S is repelled by all $i' \in \Gamma(j_S) \setminus \{\sigma(j_S)\}$ w.r.t. L_{k-1}. Hence, $\sigma'(j_S) = \sigma(j_S) = i$. With (3) it follows that $j_S \in S_i(L'_{\leq k-1}, \sigma')$. Now let $j_S \in S_i(L'_{\leq k-1}, \sigma')$. If $\sigma(j_S) = \sigma'(j_S) = i$, then as above with (3) it follows that $j_S \in S_i(L_{\leq k-1}, \sigma)$. Now assume toward contradiction $\sigma(j_S) \neq i$. By (2), j_S is not repelled by $\sigma(j_S)$ w.r.t. $L_{\leq k-1}$; By (3) this means that j_S is also not repelled by $\sigma(j_S) \neq i$ w.r.t. $L'_{\leq k-1}$. Hence, $j_S \notin S_i(L'_{\leq k-1}, \sigma')$, a contradiction. Let W_0 be the minimal $W \geq 0$ such that

$$p\left(\left\{j' \in \sigma^{-1}(i) : \frac{1}{2} < p_{j'} \leq W\right\}\right) + p(S_i(L_{\leq k-1}, \sigma)) + p_j > \left(\frac{11}{6} + 2\epsilon\right)\tau.$$

Since i repels all jobs j' with $1/2 < p_{j'} \leq W$ w.r.t. $L_{\leq k}$, we get

$$\left\{j' \in \sigma'^{-1}(i) : \frac{1}{2} < p_{j'} \leq W\right\} \subseteq \left\{j' \in \sigma^{-1}(i) : \frac{1}{2} < p_{j'} \leq W\right\}.$$

This implies that W'_0, the minimal $W \geq 0$ with

$$p\left(\left\{j' \in \sigma'^{-1}(i) : \frac{1}{2} < p_{j'} \leq W\right\}\right) + p(S_i(L'_{\leq k-1}, \sigma')) + p_j > \left(\frac{11}{6} + 3\epsilon\right)\tau.$$

is at least as big as W_0, i.e., $W'_0 \geq W_0$. This means all jobs repelled by i w.r.t. $L_{\leq k}$ are also repelled w.r.t. $L'_{\leq k}$, which implies $R(L'_{\leq k}, \sigma') \supseteq R(L_{\leq k}, \sigma)$. If equality does not hold, then $\Phi(L'_{\leq k}, \sigma') > \Phi(L_{\leq k}, \sigma)$. Otherwise (3) is fulfilled for k. If one of the jobs repelled by i is moved, then $\tilde{\mathcal{J}}(L'_{\leq k-1}) \subsetneq \tilde{\mathcal{J}}(L_{\leq k-1})$. Otherwise, equality holds and (2) follows for k. □

Theorem 50. *We can find a $(11/6+\epsilon)$-approximate solution for* RESTRICTED ASSIGNMENT *in time $n^{O(1/\epsilon \log(n))}$ for every $\epsilon > 0$, where $n = |\mathcal{J}| + |\mathcal{M}|$.*

The problem from the previous chapter has an intriguing special case where each job is allowed on two machines. This has a natural interpretation as a graph problem and will be studied in this chapter.

7 THE GRAPH BALANCING PROBLEM

In this chapter we consider weighted, undirected multigraphs that may contain loops. We write such a multigraph as $G = (V, E, r, w)$, where V is the set of vertices, E is the set of edge identities, and r is a function $E \to \{\{u, v\} : u, v \in V\}$ that defines the endpoints for every edge. Note that in the definition above we allow $u = v$, which describes a loop. E is often defined as a set of vertex pairs. We use the function r instead, since it avoids some issues due to multigraphs. The weight function $w : E \to \mathbb{R}_{>0}$ assigns positive weights to the edges. In the Graph Balancing problem we want to compute an orientation of the edges, i.e., one of the ways to turn the graph into a directed graph. The goal is to minimize the maximum weighted in-degree over all vertices, that is $\max_{v \in V} \sum_{e \in \delta^-(v)} w(e)$, where $\delta^-(v)$ are the incoming edges of vertex v in the resulting digraph. Apart from being an arguably natural problem, Graph Balancing has been of particular interest to the scheduling community. It is one of the simplest special cases of makespan minimization on unrelated machines for which an inapproximability bound of $1.5 - \epsilon$ is known, which is already the best that is known in the general problem. In the interpretation as a scheduling problem, machines correspond to vertices and jobs to edges, i.e., each job has only two potential machines to which it can be assigned. The problem was introduced by Ebenlendr, Krčál, and Sgall [28] and they gave a polynomial time 1.75-approximation for it. Indepenently and under a different name, Asahiro et al. [9] studied the problem and showed that no $(1.5 - \epsilon)$-approximation is possible unless P = NP. The algorithm by Ebenlendr et al. rounds the solution of a particular linear programming formulation. This appears to be the best one can hope for using their techniques, since the ratio between integral optimum and fractional optimum of the LP, the integrality gap, can be arbitrarily close to 1.75 [28]. Using a completely different approach to [28], Huang and Ott developed a purely combinatorial algorithm for the problem [35]. With $5/3 + 4/21 \approx 1.857$, however, their approximation ratio is inferior to the original algorithm. Another algorithm for Graph Balancing, developed by Wang and Sitters [71], achieves an approximation ratio of $11/6 \approx 1.833$, i.e., also worse than the original, but notable for being simpler. For the special case of only two different edge weights, three independent groups found a tight 1.5-approximation [20, 35, 60].

A good candidate for a stronger linear program to that from [28] is the configuration LP studied in the previous two chapters. It was introduced

by Bansal and Sviridenko for the more general problem Scheduling on Unrelated Machines and the closely related Santa Claus problem [12]. It is easy to show that this LP is at least as strong as the LP from [28] (see the same paper), i.e., the integrality gap must be at most 1.75 as well. The best lower bound known is 1.5 (see for instance [38], this holds even for the case of Graph Balancing). In recent literature, the configuration LP has enabled breakthroughs in the restricted variants for both of the problems above [8, 68]. The restricted variant of Scheduling on Unrelated Machines (also known as Restricted Assignment) can be seen as Graph Balancing with hyperedges. In particular, it contains the Graph Balancing problem as a special case. In this setting, the configuration LP was shown first to have an integrality gap of at most $33/17 \approx 1.941$ [68], which was improved to $11/6 \approx 1.833$ by us [41] (see previous chapter). In this chapter, we present a sophisticated local search algorithm for Graph Balancing and obtain the following result.

Theorem 51. *The integrality gap of the configuration LP is at most* 1.749 *for Graph Balancing.*

In other words, it is stronger than the LP from [28]. This non-constructive proof is by a local search algorithm that is not known to terminate in polynomial time. Although this does not give a polynomial time approximation algorithm, it is strong evidence that such an algorithm can be developed using the configuration LP. Furthermore, the optimal solution can be estimated in polynomial time within a factor of $1.749 + \epsilon$ for any $\epsilon > 0$ by approximating the configuration LP. We emphasize that the purpose of this theorem is to show a separation between the configuration LP and the previously used LP relaxation. The constants in the proof are not optimized. We chose to keep the case analysis (which is already difficult) and constants as simple as possible instead of improving the third decimal place. Our work in [44] indicates that earlier bounds on the integrality gap of the configuration LP in related problems disregard many constraints enforced on small edges/jobs, but that without them the LP might be much weaker. More precisely, these proofs used only properties that are already enforced by a weaker configuration LP that allows small edges/jobs to appear fractionally in a configuration. This weaker LP, however, has an integrality gap strictly higher than 1.5, whereas for the configuration LP it is still open whether 1.5 is the correct answer. The backbone of our new proof is the utilization of these small edge constraints (see end of Section 7.1). This may be relevant to other related local search based proofs as well.

OTHER RELATED WORK. The problem of minimizing the maximum out-degree is equivalent to the maximum in-degree. The very similar problem of maximizing the minimum in- or out-degree has been settled by Wiese and Verschae [70]. They gave a 2-approximation and this is the best possible assuming $P \neq NP$. Surprisingly, this holds even in the unrelated case when the value of an edge may be different on each end. They do not use the configuration LP, but it is easy to also get a bound of 2 on its integrality

gap using their ideas. For the restricted case (a special case of the unrelated one) this bound of 2 was already proven by [18]. We are not aware of any evidence that GRAPH BALANCING is easier on simple weighted graphs (without multiedges and loops). The same reduction for the state-of-the-art lower bound holds even in that case. A number of recent publications deal with the important question on how related local search algorithms can be turned into efficient algorithms [6, 7, 40, 62].

NOTATION. For some $v \in V$ we will denote by $\delta(v)$ the incident edges, i.e. those $e \in E$ with $v \in r(e)$. When a particular orientation is clear from the context, we will write $\delta^-(v)$ for the incoming edges and $\delta^+(v)$ for the outgoing edges of a vertex. For some $F \subseteq E$ we will denote by $\delta_F(v)$ the incident edges of v restricted to F and $\delta_F^-(v)$, $\delta_F^+(v)$ accordingly. For some $e \in E$ we will describe by $t(e) \in r(e)$ the vertex it is oriented towards and by $s(e) \in r(e)$ the vertex it is leaving. For a loop e, i.e., $r(e) = \{v\}$ for some $v \in V$, it always holds that $t(e) = s(e)$. For a subset of edges $S \subseteq E$ we will write $w(S)$ for $\sum_{e \in S} w(e)$ and similar for other functions over the edges.

LP RELAXATIONS. The following linear programs have no objective functions. Instead they parameterized by τ, the makespan. The optimum is the lowest τ for which it is feasible. This will be denoted by OPT* (referring to the configuration LP in the rest of the chapter). First we look at the assignment LP by Lenstra, Shmoys, and Tardos [56]. It has a variable $x_{e,v}$ for every vertex v and incident edge $e \in \delta(v)$, which indicates whether e is oriented towards v.

The assignment LP.

$$\sum_{e \in \delta(v)} w(e) \cdot x_{e,v} \leq \tau \quad \forall v \in V,$$

$$\sum_{v \in r(e)} x_{e,v} = 1 \quad \forall e \in E,$$

$$x_{e,v} \in [0,1]$$

The first constraint ensures that no vertex has more than weight τ of edges oriented towards it. The second one describes that each edge is oriented towards one vertex. The assignment LP has an integrality gap of 2. Let B denote the big edges e, which have $w(e) > 0.5 \cdot \tau$. It is clear that an integral orientation can assign at most one such edge to each vertex. Ebenlendr et al. [28] show that adding the constraint $\sum_{e \in \delta_B(v)} x_{e,v} \leq 1 \quad \forall v \in V$ improves the integrality gap to 1.75. They also give other LP relaxations, but show that none of them have an integrality gap strictly better than 1.75.

Now we will introduce the configuration LP. A configuration is a subset of edges that can be oriented towards a particular vertex without exceeding a particular makespan τ. Formally, we define the configurations of a vertex v and a makespan τ as $\mathcal{C}(v,\tau) := \{C \subseteq \delta(v) : w(C) \leq \tau\}$. The configuration

LP now assigns fractions of configurations to each machine.

Primal of the configuration LP.

$$\sum_{v \in V} \sum_{C \in \mathcal{C}(v,\tau)} x_{v,C} \leq 1 \qquad \forall v \in V$$
$$\sum_{v \in r(e)} \sum_{C \in \mathcal{C}(v,\tau): e \in C} x_{v,C} \geq 1 \qquad \forall e \in E$$
$$x_{v,C} \geq 0$$

We are particularly interested in the dual of the configuration LP (which is constructed after adding the objective function $\max (0, \ldots, 0) \cdot x$).

Dual of the configuration LP.

$$\min \sum_{v \in V} y_v - \sum_{e \in E} z_e$$
$$\text{s.t.} \sum_{e \in C} z_e \leq y_v \qquad \forall v \in V, C \in \mathcal{C}(v,\tau)$$
$$y, z \geq 0$$

Although the configuration LP has exponential size, a $(1+\epsilon)$-approximation can be computed in polynomial time for every $\epsilon > 0$ [12]. Note that τ is considered a constant in both the primal and the dual. A common idea for proving τ is lower than the optimum is to show that the dual is unbounded for τ (instead of directly showing that the primal is infeasible for τ).

Lemma 52. *If there exists $y : V \to \mathbb{R}_{\geq 0}$, and $z : E \to \mathbb{R}_{\geq 0}$ with $\sum_{e \in C} z(e) \leq y(v)$ for all $v \in V, C \in \mathcal{C}(v,\tau)$ and $\sum_{v \in V} y(v) < \sum_{e \in E} z(e)$, then $\tau < \text{OPT}^*$.*

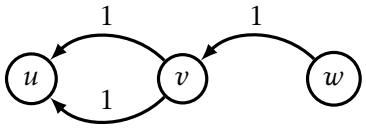
This holds because y, z is a feasible solution with negative objective value for the dual and so are the same values scaled by any $\alpha > 0$. This way an arbitrarily low objective value can be obtained. Lemma 52 can be seen as a generalization of a space argument: Consider $y(v) = \tau$ and $z(e) = w(e)$. Then for all $v \in V, C \in \mathcal{C}(v,\tau)$, $\sum_{e \in C} z(e) = \sum_{e \in C} w(e) \leq \tau = y(v)$. Thus, by the Lemma we have that $|V| \cdot \tau = \sum_{v \in V} y(v) < \sum_{e \in E} z(e) = w(E)$ implies $\tau < \text{OPT}^*$.

7.1 INFORMAL OVERVIEW

Before we give a formal definition of the local search algorithm, we devote this section to giving intuition. We start with very easy algorithms and give challenging instances that motivate the more advanced ideas.

As a toy algorithm consider the following: Start with an arbitrary orientation and repeat until all vertices are good. Good means the weighted in-degree of the vertex is at most the desired value, e.g., $1.749 \cdot \text{OPT}^*$. Whenever there

is an edge oriented towards a bad vertex such that its other vertex is good and would remain good even if the edge was flipped (this is called a valid flip), flip this edge. In the following example, both LP and integral optimum are 1. Suppose the algorithm tries to obtain a solution of makespan $2-\epsilon$ with $\epsilon > 0$.



u is bad, v and w are good. However, the algorithm cannot flip one of the edges between u and v, because this would make v bad. It will not flip the edge between v and w, because v is already good. Hence, the algorithm fails. Obviously, in this example we should flip the edge between v and w and then fix u. Let us try to integrate this in the algorithm. We introduce the concept of *pending flips*. When the algorithm wants to flip an edge, but it cannot, because this would make a vertex bad, we add this edge to a list of pending flips. These will be executed once the flip is valid. In the example above, the algorithm could add the edges between u and v to the pending flips, but would not change their orientation, yet. For a pending flip e let us call $s(e)$ the *prospect vertex* and $t(e)$ the *current vertex*. As seen in the example above a sensible local search algorithm should try to move edges away from pending flips' prospect vertices as well (in addition to the bad vertices).

We now state the second toy algorithm. Initialize the pending flips as an empty list. Repeat the following until all vertices are good. Let U be the set of vertices that are either bad or prospect vertices of pending flips. Find an edge from V/U to U and add it to the pending flips. As long as there is a valid pending flip, (1) execute it and (2) delete all pending flips added after it. (2) is to ensure obsolete pending flips are removed. Without it there can be situations where a pending flip has its current vertex in $V \setminus U$, because the pending flip that initially led us to adding it has been executed. This algorithm always succeeds for makespan 1.75 in the special case where weights are in $(0, 0.5] \cup \{1\}$ and $\mathrm{OPT}^* = 1$. We will quickly go over the arguments, since it gives a good idea on how to use the dual of the configuration LP. At this point we will only argue that the algorithm does not get stuck. Normally, we would also have to prove that it terminates (we omit this for sake of brevity). Suppose toward contradiction that the algorithm gets stuck, i.e., there is no valid pending flip and there are no edges from V/U to U. Let F be the big edges e with $t(e) \in U$. We set

$$z(e) = \begin{cases} w(e) & \text{if } t(e) \in U, \\ 0 & \text{otherwise.} \end{cases}$$

$$y(v) = \begin{cases} w(\delta^-(v)) + \frac{1}{4}|\delta_F^+(v)| - \frac{1}{4}|\delta_F^-(v)| & \text{if } v \in U \text{ is good,} \\ w(\delta^-(v)) + \frac{1}{4}|\delta_F^+(v)| - \frac{1}{4}|\delta_F^-(v)| - 0.1 & \text{if } v \in U \text{ is bad,} \\ \frac{1}{4}|\delta_F^+(v)| - \frac{1}{4}|\delta_F^-(v)| & \text{if } v \in V \setminus U. \end{cases}$$

What is left to do is to check that for these values the premise of Lemma 52 is fulfilled. Let $v \in V$ and $C \in \mathcal{C}(v,1)$. Recall that by definition, $C \subseteq \delta(v)$ and $w(C) \leq 1$. We have to verify that $z(C) \leq y(v)$.

Case 1: $v \in V \setminus U$. Since the algorithm is stuck, there is no edge $e \in \delta^+(v)$ with $t(e) \in U$. Hence, $z(e) = 0$ for all $e \in \delta(v)$ and $z(C) = 0$. By definition of F we have that $\delta_F^-(v) = \emptyset$. Thus, $y(v) \geq 0 = z(C)$.

Case 2: $v \in U$. If v is bad, then $w(\delta^-(v)) > 1.75$ and

$$y(v) \geq w(\delta_F^-(v)) - \frac{1}{4}|\delta_F^-(v)| - 0.1 \geq \frac{3}{4}|\delta_F^-(v)| - 0.1 \geq 1.4 > z(C),$$

if $|\delta_F^-(v)| \geq 2$. Otherwise, $|\delta_F^-(v)| \leq 1$ and thus,

$$y(v) \geq w(\delta^-(v)) - \frac{1}{4}|\delta_F^-(v)| - 0.1 > 1.75 - \frac{1}{4} - 0.1 \geq 1.4 > z(C).$$

If $v \in U$ is good, then it is the prospect vertex of some pending flip e. An invariant of the algorithm is that all pending flips have their current vertex in U. In particular, $z(e) = w(e)$. Furthermore, e is not a valid flip. Thus, $w(\delta^-(v)) + w(e) > 1.75$. Also note that $|\delta_F^-(v)| \leq 1$, since v is good. If $w(e) \leq 0.5$, then

$$y(v) \geq w(\delta^-(v)) - \frac{1}{4}|\delta_F^-(v)| > 1.75 - w(e) - \frac{1}{4} \geq 1 \geq z(C).$$

If $w(e) = 1$, then $e \in \delta_F^+(v)$. Hence, if $|\delta_F^-(v)| = 0$,

$$y(v) \geq w(\delta^-(v)) + \frac{1}{4}|\delta_F^+(v)| - \frac{1}{4}|\delta_F^-(v)| \geq 1.75 - w(e) + \frac{1}{4} \geq 1 \geq z(C).$$

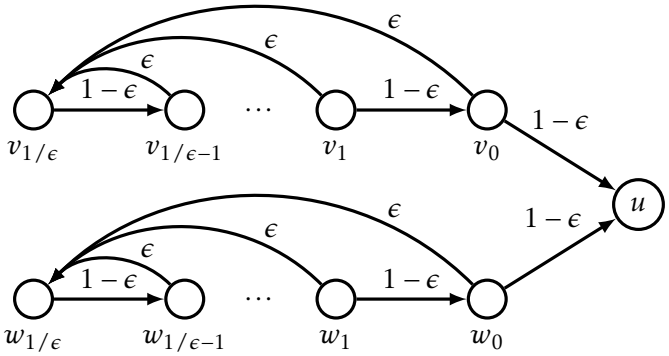
If on the other hand $|\delta_F^-(v)| = 1$,

$$y(v) \geq w(\delta_F^-(v)) + \frac{1}{4}|\delta_F^+(v)| - \frac{1}{4}|\delta_F^-(v)| \geq 1 + \frac{1}{4} - \frac{1}{4} \geq 1 \geq z(C).$$

The second condition of Lemma 52 is that $z(E) > y(V)$. This holds because

$$z(E) = \sum_{v \in U} w(\delta^-(v)) = \sum_{v \in U} w(\delta^-(v)) + \frac{1}{4} \sum_{v \in V} [|\delta_F^+(v)| - |\delta_F^-(v)|] > y(V).$$

The strict inequality follows from the fact that there is at least one bad vertex. In the general case this algorithm does not get better than 2 as can be seen in the example below. In a similar, but more complicated way one can also show that in the special case with weights in $(0, 0.5] \cup \{1\}$, the algorithm does not succeed for $1.75 - \epsilon$, where $\epsilon > 0$ is arbitrary. In other words, the analysis for 1.75 is tight.



The LP and integral optima are again 1. Suppose the algorithm tries to find a solution with makespan $2-3\epsilon$. The only bad vertex is u. Hence, the algorithm will add the $(1-\epsilon)$-edges to the pending flips one after another. However, at the time it reaches $v_{1/\epsilon}$ or $w_{1/\epsilon}$ it will get stuck, since there is a load of $1/\epsilon \times \epsilon$ on them. The way to fix this is to allow edges (in this case those of weight ϵ) to also be flipped towards vertices in U, i.e., vertices where we wanted to reduce the load. We cannot simply allow arbitrary flips back and forth between U, because we have to take care that the algorithm eventually terminates. This is where the concept of vertices *repelling* edges comes in: Depending on the current orientation of the edges and the list of pending flips we will define a binary relation between vertices and edges. The exact definition has to be chosen carefully. For a pair (v,e) of this relation we write vertex v repels edge e. When a vertex repels an edge, it means it is undesirable that the edge is oriented towards this vertex. When it does not repel the edge, we do not care. This will be used in the algorithm when adding pending flips: An edge e may only be added to the pending flips, when it is repelled by its current vertex and not repelled by its prospect vertex.

In the earlier toy algorithms a vertex either repels all edges (when it is in U) or none (when it is not in U). By adding a fine-grained strategy of repelled edges, we gain much more flexibility. A strategy that appears particularly simple and powerful is the following. (1) We let bad vertices repel all edges. Moreover, (2) for every pending flip e with prospect vertex v we find maximum threshold W such that all edges in $\delta^-(v)$ with weight at least W are already enough to prevent the flip from being executed, i.e., their total weight is greater than $1+R-w(e)$. We let v repel all edges of weight at least $\min\{w(e), W\}$.

Now the third toy algorithm is to repeat the following until all vertices are good. Find an edge e that is repelled by $t(e)$, but not by $s(e)$ and add it to the list of pending flips. As long as there is a valid pending flip, execute it and delete all pending flips added after it. This algorithm succeeds in the previous example. It will add the $(1-\epsilon)$-edges to the pending flips, but the vertices $u, v_1, \ldots, v_{1/\epsilon-1}, w_1, \ldots, w_{1/\epsilon-1}$ repel only the $(1-\epsilon)$-edges and not the ϵ-edges. Once the pending flips reach $w_{1/\epsilon}$ or $v_{1/\epsilon}$, these vertices will repel the ϵ-edges and start flipping them. Finally, there will be enough space on $w_{1/\epsilon}$ or $v_{1/\epsilon}$ and the $1-\epsilon$ edges can be flipped one after another.

For the special case described earlier (where $w(e) \in (0, 0.5] \cup \{1\}$) this is very close to the algorithm that gives us the bound of 1.74. However, to make the analysis work, we add a couple of tweaks. One of them is the idea of critical vertices. When the threshold of repelled edges, i.e., $\min\{w(e), W\}$, reaches a low value, e.g., 0.26, we make the vertex repel all edges. We call such a vertex a *critical* vertex. This tweak (together with some technicalities) allows us to argue that at least half of the critical vertices are prospect vertices of pending flips for tiny edges. This is helpful, since (unless the pending flip is valid) such prospect vertices have a lot of weight oriented towards them and this makes it easier to argue that the configuration LP also cannot distribute this weight very well; thereby reaching a contradiction.

An important novelty in this work compared to earlier local search algorithms is that we are able to repel only a subset of edges small/tiny edges. In the Restricted Assignment proofs [68, 41] it is always the case that a machine (vertex) repels all small/tiny jobs (edges) or none. To utilize the flexibility in the small/tiny edges we scale the z values of tiny edges up by a factor $\beta > 1$. It is not obvious at all that this works out and the proof is quite tricky.

In a sense, we push the threshold for repelling all edges down to some value less than 0.5 (see description of critical vertices above). A logical conclusion to draw would be that pushing it down even more (or removing it completely) should give an even better algorithm. This would bring us back to toy algorithm 3. In fact, we are not aware of bad instances. It is an intriguing question whether this can also be proved that this simple algorithm is good.

7.2 GRAPH BALANCING IN A SPECIAL CASE

To introduce our techniques, we first consider a simplified case where $w(e) \in (0, 0.5] \cup \{1\}$ for each $e \in E$ and the configuration LP is feasible for 1. We will show that there exists an orientation with maximum weighted in-degree $1 + R$ where $R = 0.74$.

Definition 11 (Tiny, small, big edges)**.** *We call an edge e* tiny*, if $w(e) \leq 1 - R$;* small*, if $1 - R < w(e) \leq 1/2$; and* big*, if $w(e) = 1$. We will write for the tiny, small, and big edges $T \subseteq E$, $S \subseteq E$, and $B \subseteq E$, respectively.*

Definition 12 (Good and bad vertices)**.** *For a given orientation, we call a vertex v* good*, if $w(\delta^-(v)) \leq 1 + R$. A vertex is* bad*, if it is not good.*

The local search algorithm starts with an arbitrary orientation and flips edges until all vertices are good. During this process, a vertex that is already good will never be made bad. It is then proved that (1) the algorithm terminates and (2) when it cannot find a useful edge to flip, the configuration LP also cannot distribute the edges well, i.e., $\text{OPT}^* > 1$, a contradiction.

7.2.1 *Algorithm*

The central data structure we use is an ordered list of pending flips $P = (e_1^P, e_2^P, \dots, e_\ell^P)$. Here, every component e_k^P, stands for an edge the algorithm wants to flip. If P is clear from the context, we simply write *flip* when we speak of a pending flip e_k^P. A *tiny flip* is a flip where e_k^P is tiny. In the same way we define *small* and *big flips*. The prospect vertex of a flip e_k^P is the vertex $s(e_k^P)$ to which we want to orient it. The algorithm will not perform the flip, if this would create a bad vertex, i.e., $w(\delta^-(s(e_k^P))) + w(e_k^P) > 1 + R$. If it does not create a bad vertex, we say that the flip e_k^P is a *valid flip*. For every $0 \le k \le \ell$ define $P_{\le k} := (e_1^P, \dots, e_k^P)$, i.e., the first k elements of P (with $P_{\le 0}$ being the empty list).

At each point during the execution of the algorithm, the vertices repel certain edges. This can be thought of as a binary relation between vertices and their incident edges, i.e., a subset of $\{(v,e) : v \in r(e)\}$, and this relation changes dynamically as the current orientation or P change. The definition of which vertices repel which edges is given later. The algorithm will only add a new pending flip e to P, if e is repelled by the vertex it is oriented towards and not repelled by the other.

Input: Weighted multigraph $G = (V, E, r, w)$ with $\mathrm{OPT}^* = 1$ and $w(e) \in (0, 0.5] \cup \{1\}$ for all $e \in E$
Result: Orientation $s, t : E \to V$ with maximum weighted in-degree $1 + R$
let $s, t : E \to V$ map arbitrary source and target vertices to each edge ;
// i.e., $\{s(e), t(e)\} = r(e)$ for all $e \in E$
$\ell \leftarrow 0$; // number of pending edges P to flip
while *there is a bad vertex* **do**
  if *there exists a valid flip* $e \in P$ **then**
    let $0 \le k \le \ell$ be minimal such that e is repelled by $t(e)$ w.r.t. $P_{\le k}$;
    exchange $s(e)$ and $t(e)$;
    delete $P_{k+1}, \dots, P_\ell$; $\ell \leftarrow k$;
  else
    choose an edge $e \in E \setminus P$ with $w(e)$ minimal and e is repelled by $t(e)$ and not repelled by $s(e)$ w.r.t. P ;
    $P_{\ell+1} \leftarrow e$; $\ell = \ell + 1$; // Append e to P

Algorithm 6: Local search algorithm for simplified GRAPH BALANCING

REPELLED EDGES. Consider the current list of ℓ pending flips $P_{\le \ell}$. The repelled edges are defined inductively. For some $k \le \ell$ we will now define the repelled edges w.r.t. $P_{\le k}$.

(INITIALIZATION) If $k = 0$, let every bad vertex v repel every edge in $\delta(v)$. Furthermore, let every vertex v repel every loop e where $\{v\} = r(e)$.

(MONOTONICITY) If $k > 0$ and v repels e w.r.t. $P_{\leq k-1}$, then let v repel e w.r.t. $P_{\leq k}$.

The rule on loops is only for a technical reasons. Loops will never appear in the list of pending flips. The remaining rules regard $k > 0$ and the last pending flip in $P_{\leq k}$, e_k^P. The algorithm should reduce the load on $s(e_k^P)$ to make it valid.

Which edges exactly does the prospect vertex of e_k^P, i.e., $s(e_k^P)$, repel? First, we define $\tilde{E}(P_{\leq k-1}) \subseteq E$ where $e \in \tilde{E}(P_{\leq k-1})$ if and only if e is repelled by $s(e)$ w.r.t. $P_{\leq k-1}$. We will omit $P_{\leq k-1}$ when it is clear from the context. $\tilde{E}$ are edges that we do not expect to be able to flip: Recall that when an edge e is repelled by $s(e)$, it cannot be added to P. Moreover, for every W define $E_{\geq W} = \{e \in E : w(e) \geq W\}$. We are interested in values W such that

$$w(\delta^-_{\tilde{E} \cup E_{\geq W}}(s(e_k^P))) + w(e_k^P) > 1 + R. \tag{13}$$

Let $W_0 \in (0, w(e_k^P)]$ be maximal such that (13) holds. To be well-defined, we set $W_0 = 0$, if no such W exists. In that case, however, it holds that $w(\delta^-(s(e_k^P))) + w(e_k^P) \leq 1 + R$. This means that e_k^P is valid and the algorithm will remove it from the list immediately. Hence, the case is not particularly interesting. We define the following edges to be repelled by $s(e_k^P)$:

(UNCRITICAL) If $W_0 > 1 - R$, let $s(e_k^P)$ repel every edge in $\delta_{\tilde{E} \cup E_{\geq W_0}}(s(e_k^P))$.

(CRITICAL) If $W_0 \leq 1 - R$, let $s(e_k^P)$ repel every edge in $\delta(s(e_k^P))$.

Note that in the cases above there is not a restriction to incoming edges like in (13).

Fact 53. *$s(e_k^P)$ repels e_k^P w.r.t. $P_{\leq k}$.*

This is because of $W_0 \leq w(e_k^P)$ in the rules for $P_{\leq k}$. An important observation is that repelled edges are stable under the following operation.

Fact 54. *Let $e \notin P_{\leq k}$ be an edge that is not repelled by any vertex w.r.t. $P_{\leq k-1}$, and possibly by $t(e)$ (but not $s(e)$) w.r.t. $P_{\leq k}$. If the orientation of e changes and this does not affect the sets of good and bad vertices, the edges repelled by some vertex w.r.t. $P_{\leq k}$ will still be repelled after the change.*

Proof. We first argue that the repelled edges w.r.t. $P_{\leq k'}$, $k' = 0, \dots, k-1$ have not changed. This argument is by induction. Since the good and bad vertices do not change, the edges repelled w.r.t. $P_{\leq 0}$ do not change. Let $k' \in \{1, \dots, k-1\}$ and assume that the edges repelled w.r.t. $k' - 1$ have not changed. Moreover, let W_0 be as in the definition of repelled edges w.r.t. $P_{\leq k'}$ before the change. We have to understand that e is not and was not in $\delta^-_{\tilde{E} \cup E_{\geq W_0}}(s(e_{k'}^P))$ (with $\tilde{E} = \tilde{E}(P_{\leq k'-1})$). This means that flipping it does not affect the choice of W_0 and, in particular, not the repelled edges. Let v and v' denote the vertex e is oriented towards before the flip and after the flip, respectively. Since e was not repelled by v and v' w.r.t. $P_{\leq k'}$, it holds that $v, v' \neq s(e_k^P)$ or $w(e) < W_0$:

$s(e_k^P)$ repelled all edges greater or equal W_0 in both the case (uncritial) and (critical), but e was not repelled by v or v'.

Therefore $e \notin \delta^-_{E_{\geq W_0}}(s(e_{k'}^P))$ before and after the change. Moreover, since e was not repelled by any vertex w.r.t. $P_{\leq k'-1}$ (and by induction hypothesis, it still is not), it follows that $e \notin \tilde{E}(P_{\leq k'-1})$. By this induction we have that edges repelled w.r.t. $P_{\leq k-1}$ have not changed and by the same argument as before, e is not in $\delta^-_{\tilde{E}\cup E_{\geq W_0}}(s(e_k^P))$ after the change. This means W_0 has not increased and edges repelled w.r.t. $P_{\leq k}$ are still repelled. It could be that W_0 decreases, if e was in $\delta^-_{\tilde{E}\cup E_{\geq W_0}}(s(e_k^P))$ before the flip. This would mean that the number of edges repelled by $s(e_k^P)$ increases. □

We note that W_0 (in the definition of $P_{\leq k}$) is either equal to $w(e_k^P)$ or it is the maximal value for which (13) holds. Furthermore,

Fact 55. *Let W_0 be as in the definition of repelled edges w.r.t. $P_{\leq k}$. If $W_0 < w(e_k^P)$, then there is an edge of weight exactly W_0 in $\delta^-(s(e_k^P))$. Furthermore, it is not a loop and it is not repelled by its other vertex, i.e., not $s(e_k^P)$, w.r.t. $P_{\leq k}$.*

Proof. We prove this for $P_{\leq k-1}$. Since the change from $P_{\leq k-1}$ to $P_{\leq k}$ only affects $s(e_k^P)$, this suffices. All edges that are repelled by their other vertex w.r.t. $P_{\leq k-1}$ (in particular, loops) are in $\tilde{E}$. Recall that $w(\delta^-_{\tilde{E}\cup E_{\geq W_0}}(s(e_k^P))) + w(e_k^P) > 1 + R$. Assume toward contradiction there is no edge of weight W_0 in $\delta^-(s(e_k^P))$, which is not in $\tilde{E}$. This means there is some $\epsilon > 0$ such that $\delta^-_{\tilde{E}\cup E_{\geq W_0+\epsilon}}(s(e_k^P)) = \delta^-_{\tilde{E}\cup E_{\geq W_0}}(s(e_k^P))$. Hence, W_0 is not maximal. □

7.2.2 *Analysis*

The following analysis holds for the values $R = 0.74$ and $\beta = 1.1$. β is a central parameter in the proof. These can be slightly improved, but we refrain from this for the sake of simplicity. The proof consists of two parts. We need to show that the algorithm terminates and that there is always either a valid flip in P or some edge can be added to P.

Lemma 56. *The algorithm terminates after finitely many iterations of the main loop.*

Proof. We consider the potential function

$$s(P) = (g, |\tilde{E}(P_{\leq 0})|, \ldots, |\tilde{E}(P_{\leq \ell})|, -1),$$

where g is the number of good vertices. We will argue that this vector increases lexicographically after every iteration of the main loop. Intuitively $|\tilde{E}(P_{\leq k})|$, $k \leq \ell$, is an measure for progress. When an edge e is repelled by a vertex v, then we want that $s(e) = v$. $|\tilde{E}(P_{\leq k})|$ counts exactly these situations. Since ℓ is bounded by $|E|$, there can be at most $|V| \cdot |E|^{O(|E|)}$ possible values for the vector. Thus, the algorithm must terminate after at most this many iterations. In an iteration either a new flip is added to P or a flip is executed.

If a flip e is added as the $(\ell+1)$-th element of P, then clearly $\tilde{E}(P_{\leq i})$ does not change for $i \leq \ell$. Furthermore, the last component of the vector is replaced by some non-negative value.

Now consider a flip $e \in P$ that is executed. If this flip turns a bad vertex good, we are done. Hence, assume otherwise and let v and v' be the vertex it was previously and the one it is now oriented towards. Furthermore, let ℓ' be the length of P after the flip. Recall that ℓ' was chosen such that before the flip is executed v repels e w.r.t. $P_{\leq \ell'}$, but not w.r.t. $P_{\leq k}$ for any $k \leq \ell' - 1$. Also, e is not repelled by v' w.r.t. $P_{\leq k}$ for any $k \leq \ell'$ or else the flip would not have been added to P in the first place. By Fact 54 this means that repelled edges w.r.t. $P_{\leq k}$, $k \leq \ell'$, are still repelled after the flip. Because of this and because the only edge that changed direction, e, is not in $\tilde{E}(P_{\leq k})$ for any $k \leq \ell'$, $|\tilde{E}(P_{\leq k})|$ has not decreased. Finally, e has not been in $\tilde{E}(P_{\leq \ell'})$ before the flip, but now is. Thus, the first $\ell' - 1$ components of the vector have not decreased and the ℓ'-th one has increased. □

Lemma 57. *If there at least one bad vertex remaining, then there is either a valid flip in P or a flip that can be added to P.*

Proof. We assume toward contradiction that there exists a bad vertex, no valid pending flip and no edge that can be added to P. We will show that this implies $\mathrm{OPT}^* > 1$.

Like above we denote by $\tilde{E} = \tilde{E}(P)$ those edges e that are repelled by $s(e)$. In particular, if an edge e is in P or repelled by $t(e)$ it must also be in $\tilde{E}$: If such an edge is in P, this follows from Fact 53. Otherwise, such an edge must be repelled also by $s(e)$ or else it could be added to P. For every $e \in E \setminus \tilde{E}$, set $z(e) = 0$. For every $e \in \tilde{E}$, set

$$z(e) = \begin{cases} 1 & \text{if } w(e) = 1, \\ w(e) & \text{if } 1 - R < w(e) \leq 1/2, \text{ and} \\ \beta w(e) & \text{if } w(e) \leq 1 - R. \end{cases}$$

In general, we would like to set each $y(v)$ to $z(\delta^-(v))$ (or equivalently, $z(\delta^-_{\tilde{E}}(v))$). However, there are two kinds of amortization between vertices that we include in the values of y.

As can be seen in the definition of repelled edges, a vertex v repels either edges with weight at least a certain threshold $W > 1 - R$ and edges in $\delta^-(v)$ that are repelled by their other vertex; or they repel all edges. We will call vertices of the latter kind *critical*. In other words, a vertex v is critical, if in the inductive definition of repelled edges at some point $v = s(e_k^P)$ and the rule (critical) applies. There might be a coincidence where only rule (uncritical) applies for v, but this already covers all edges in $\delta^-(v)$. This is not a critical vertex. We note that every vertex that is prospect vertex of a tiny flip e_k^P must be critical: In the inductive definition when considering e_k^P we have that $W_0 \leq w(e_k^P) \leq 1 - R$ by definition.

CRITICAL VERTEX AMORTIZATION. If v is a good vertex and critical, but is not a prospect vertex of a tiny flip, set $a_v := \beta - 1$. If v is a good

vertex and prospect vertex of tiny flip (in particular, v is critical), set $a_v := -(\beta - 1)$. Otherwise, set $a_v = 0$.

BIG EDGE AMORTIZATION. Let $F \subseteq P$ denote the set of big flips. Then in particular $F \subseteq \tilde{E}$ (Fact 53). We define for all $v \in V$, $b_v := (|\delta_F^+(v)| - |\delta_F^-(v)|) \cdot (1 - R)$.

We conclude the definition of y by setting $y(v) = z(\delta^-(v)) + a_v + b_v$ for all good vertices v and $y(v) = z(\delta^-(v)) + a_v + b_v - \mu$ for all bad vertices v, where $\mu = 0.01$.

Claim 58. *It holds that* $\sum_{v \in V} y(v) < \sum_{e \in E} z(e)$.

Claim 59. $y(v), z(e) \geq 0$ *f.a.* $v \in V, e \in E$ *and f.a.* $v \in V$, $C \in \mathcal{C}(v, 1)$, $\sum_{e \in C} z(e) \leq y(v)$.

By Lemma 52 this implies that $\mathrm{OPT}^* > 1$. □

Proof of Claim 58. First we note that

$$\sum_{v \in V} b_v = (1 - R) \underbrace{\left(\sum_{v \in V} |\delta_F^+(v)| - \sum_{v \in V} |\delta_F^-(v)|\right)}_{=0} = 0.$$

Moreover, we have that $\sum_{v \in V} a_v \leq 0$: This is because at least half of all good vertices that are critical are prospect vertices of tiny flips. The proof for this is omitted due to space constraints. We conclude,

$$\sum_{e \in E} z(e) \geq \sum_{v \in V} \sum_{e \in \delta^-(v)} z(e) + \sum_{v \in V} [b_v + a_v] > \sum_{v \in V} y(v),$$

where the strict inequality holds because there exists at least one bad vertex. □

Proof of Claim 59. Let $v \in V$ and $C \in \mathcal{C}(v, 1)$. We need to show that $z(C) \leq y(v)$. Obviously the z values are non-negative. By showing the inequality above, we also get that the y values are non-negative. First, we will state some auxiliary facts.

Fact 60. *For every edge flip* $e \in P$, *we have* $w(\delta_{\tilde{E}}^-(s(e))) + w(e) > 1 + R$.

This is due to the fact that e is not a valid flip and by definition of repelled edges.

Fact 61. *If* v *is a good vertex and not prospect vertex of a tiny pending flip, then* $y(v) \geq z(C) + w(\delta_{\tilde{E}}^-(v)) - 1 + b_v$. *In other words, it is sufficient to show that* $w(\delta_{\tilde{E}}^-(v)) + b_v \geq 1$.

If v is critical, then $y(v) = z(\delta^-(v)) + \beta - 1 + b_v \geq w(\delta_{\tilde{E}}^-(v)) + z(C) - 1 + b_v$. If v is uncritical, all edges $e \in C$ with $z(e) > w(e)$ must be in $\delta^-(v)$. Otherwise, the flip e could be added to P. Therefore,

$$\begin{aligned} y(v) &= z(\delta^-(v)) + a_v + b_v \geq z(\delta_{\tilde{E}}^-(v)) - w(\delta_{\tilde{E}}^-(v)) + w(\delta_{\tilde{E}}^-(v)) + b_v \\ &\geq z(\tilde{C}) - w(\tilde{C}) + w(\delta_{\tilde{E}}^-(v)) + b_v \geq z(C) + w(\delta_{\tilde{E}}^-(v)) - 1 + b_v. \end{aligned}$$

Fact 62. *For every vertex v it holds that (1) v repels no edges, (2) v repels all edges (critical), or (3) there exists a threshold $W > 1 - R$ such that v repels edges $e \in \delta^-(v)$ if $w(e) \geq W$ or they are also repelled by $s(e)$. Furthermore, in (3) we have that $W = \min_{e \in \delta_P^+(v)} w(e)$ or $W < \min_{e \in \delta_P^+(v)} w(e)$ and $W \in \{w(e) : e \in \delta_{\tilde{E}}^-(v)\}$ (see Fact 55).*

CASE 1: v IS A BAD VERTEX. If $|\delta_F^-(v)| \geq 2$,

$$\begin{aligned} y(v) &\geq z(\delta^-(v)) - |\delta_F^-(v)| \cdot (1-R) - \mu \\ &\geq |\delta_F^-(v)| \cdot (1-(1-R)) - \mu \geq 2R - \mu \geq \beta \geq z(C). \end{aligned}$$

Otherwise, $|\delta_F^-(v)| \leq 1$ and therefore

$$\begin{aligned} y(v) &\geq z(\delta^-(v)) - 1 + R - \mu \geq w(\delta^-(v)) - 1 + R - \mu \\ &> 1 + R - 1 + R - \mu = 2R - \mu \geq \beta \geq z(C). \end{aligned}$$

Here we use that $w(\delta^-(v)) > 1 + R$ by the definition of a bad vertex. Assume for the remainder that v is a good vertex, in particular that $|\delta_F^-(v)| \leq 1$.

CASE 2: v IS GOOD AND PROSPECT VERTEX OF A TINY FLIP. Using Fact 60 we get $w(\delta_{\tilde{E}}^-(v)) > 1 + R - (1-R) = 2R$. Since $z(e) \geq w(e)$ for all $e \in \tilde{E}$, we also have $z(\delta^-(v)) \geq 2R$. Moreover, since the vertex is good, $|\delta_F^-(v)| \leq 1$ and consequently $b_v \geq -(1-R)$. We conclude

$$\begin{aligned} y(v) &\geq z(\delta^-(v)) - (\beta - 1) - (1-R) \geq 2R - (\beta - 1) - (1-R) \\ &= \beta + \underbrace{3R - 2\beta}_{\geq 0} \geq z(C). \end{aligned}$$

CASE 3: v IS GOOD, NOT PROSPECT VERTEX OF A TINY FLIP, BUT PROSPECT VERTEX OF A SMALL FLIP. If $|\delta_F^-(v)| = 0$, again with Fact 60 we obtain

$$w(\delta_{\tilde{E}}^-(v)) + b_v \geq 1 + R - 0.5 + 0 > 1.$$

This suffices because of Fact 61. Moreover, if $|\delta_F^-(v)| = 1$ and $\delta_{\tilde{E}}^-(v)$ contains a small edge, it holds that

$$\begin{aligned} w(\delta_{\tilde{E}}^-(v)) + b_v &\geq w(\delta_F^-(v)) + (1-R) + b_v \\ &\geq 1 + (1-R) - (1-R) = 1. \end{aligned}$$

Here we use that the small edge must have a size of at least $1 - R$. The case that remains is where $\delta_{\tilde{E}}^-(v)$ contains one edge from F and tiny edges. Since the overal weight of $\delta_{\tilde{E}}^-(v)$ is at least $1 + R - 0.5$, the weight of tiny edges is at least $R - 0.5$. Thus, the z-value of the tiny edges is at least $\beta(R - 0.5)$. If v is critical,

$$\begin{aligned} y(v) &\geq z(\delta^-(v)) - (1-R) + (\beta - 1) \\ &\geq 1 + \underbrace{\beta(R - 0.5) - (1-R)}_{\geq 0} + (\beta - 1) \geq z(C). \end{aligned}$$

Notice that $\beta(R-0.5) \geq 1-R$ by choice of R and β. Assume for the remainder of this case that v is uncritical. If C contains a big edge, this must be the only element in C. Therefore,

$$y(v) \geq z(\delta^-(v)) - (1-R) \geq 1 + \beta(R-0.5) - (1-R) \geq 1 = z(C).$$

Consider the case where $C \cap \tilde{E}$ contains at most one small edge and no big edge. Since v is uncritical, all tiny edges in C with positive z value must also be in $\delta^-(v)$. Therefore, $z(C) \leq 0.5 + z(\delta_T^-(v))$. Thus,

$$\begin{aligned} y(v) \geq z(\delta^-(v)) - (1-R) &\geq 1 + z(\delta_T^-(v)) - (1-R) \\ &> 0.5 + z(\delta_T^-(v)) \geq z(C). \end{aligned}$$

In the final case $C \cap \tilde{E}$ contains $k \in \{2,3\}$ small edges. We let s denote the weight of the smallest edge with a pending flip whose prospect vertex is v. Since v is not critical and there is no small edge in $\delta^-(v)$, v repels exactly those edges with weight at least s. This means the small edges in $C \cap \tilde{E}$ must be of weight at least s: Otherwise, they could be added as pending flips. Therefore,

$$\begin{aligned} y(v) &\geq z(\delta^-(v)) - (1-R) \geq z(\delta_F^-(v)) + z(\delta_T^-(v)) - (1-R) \\ &\geq 1 + \beta(w(\delta_{\tilde{E}}^-(v)) - w(\delta_F^-(v))) - (1-R) = R + \beta(w(\delta_{\tilde{E}}^-(v)) - 1) \\ &\geq R + \beta(R-s). \end{aligned}$$

Note that $s \leq 1/k$ and, by choice of β, $k - \beta(k-1) \geq 0$. It follows that

$$\begin{aligned} z(C) &\leq ks + \beta(1-ks) \\ &\leq y(v) + ks - R + \beta(1-R-(k-1)s) \leq y(v) + k\frac{1}{k} - R + \beta\left(1 - R - \frac{k-1}{k}\right) \\ &= y(v) + 1 - R + \beta\left(\frac{1}{k} - R\right) \leq y(v) + 1 - R + \beta(0.5 - R) \leq y(v). \end{aligned}$$

CASE 4: v IS GOOD AND NOT PROSPECT VERTEX OF A SMALL/TINY FLIP. If $|\delta_F^+(v)| = 0$, then v does not repel any edges. In particular, $\delta_F^-(v) = \emptyset$ and every $e \in \delta(v)$ with $z(e) > 0$ must be in $\delta^-(v)$. Therefore, $z(C) \leq z(\delta^-(v)) \leq y(v)$. We assume in the remainder that $|\delta_F^+(v)| = 1$.

If $|\delta_F^-(v)| = 1$, we get $w(\delta_{\tilde{E}}^-(v)) + b_v \geq 1 + 0$. Otherwise, it must hold that $|\delta_F^-(v)| = 0$ and $w(\delta_{\tilde{E}}^-(v)) + b_v \geq R + (1-R) = 1$. □

Proof of Claim 58. First we note that

$$\sum_{v \in V} b_v = (1-R) \underbrace{\left(\sum_{v \in V} |\delta_F^+(v)| - \sum_{v \in V} |\delta_F^-(v)|\right)}_{=0} = 0.$$

Moreover, we have that $\sum_{v \in V} a_v \leq 0$: This is because at least half of all good vertices that are critical are prospect vertices of tiny flips. When a vertex v is critical but not prospect vertex of a tiny flip, there must be a non-tiny flip

e_k^P with $s(e_k^P) = v$ such that in the definition of repelled edges for $P_{\leq k}$ we have $W_0 \leq 1 - R$. By Fact 55 there exists an edge of weight W_0 (hence, tiny) in $\delta^-(v)$ which is not repelled by its other vertex $v' \neq v$ w.r.t. $P_{\leq k}$. We argue later that this edge has already been in $\delta^-(v)$ when the list of pending flips has consisted only of the k flips in $P_{\leq k}$. Since a tiny flip will always be added before the next non-tiny flip (edges of minimal weight are chosen), we know that e_{k+1}^P must be a tiny flip and $s(e_{k+1}^P)$ is the prospect vertex of a tiny flip. Also note that no vertex can be prospect vertex of two tiny flips, since after adding one such flip the vertex repels all edges.

The reason why the edge has already been in $\delta^-(s(e_k^P))$ is the following. When an edge e is flipped towards $v = s(e_k^P)$ and it is repelled by v w.r.t. $P_{\leq k}$, then e must have been a flip in P earlier in the list than e_k^P. This means that at least $e_k^P, \ldots, e_\ell^P$ are removed from P. Because e_k^P is still in P, we can therefore assume that this has not happened. If W_0 in the definition of the repelled edges w.r.t. $P_{\leq k}$ has not decreased since the last time the list consisted only of $P_{\leq k}$, we are done, since this would mean the edge of weight W_0 has been repelled by v all this time and could not have been added to $\delta^-(v)$. This is indeed the case. W_0 can only decrease when an edge from $\delta^-_{\tilde{E} \cup E_{\geq W_0}}(v)$ is flipped. This, on the other hand, causes at least $e_{k+1}^P, \ldots, e_\ell^P$ to be removed from P. This finished the proof of $\sum_{v \in V} a_v \leq 0$. We conclude,

$$\sum_{e \in E} z(e) \geq \sum_{v \in V} \sum_{e \in \delta^-(v)} z(e) + \underbrace{\sum_{v \in V} [b_v + a_v]}_{\leq 0} > \sum_{v \in V} y(v),$$

where the strict inequality holds because of the definition of $y(v)$ and because there exists at least one bad vertex. □

7.3 GRAPH BALANCING IN THE GENERAL CASE

We can still assume w.l.o.g. that $\mathrm{OPT}^* = 1$ and therefore $w(e) \in (0, 1]$ for every $e \in E$. If this does not hold, we can simply scale every edge weight by $1/\mathrm{OPT}^*$. We extend the definition of tiny, small, and big edges, but this time we set the threshold for tiny edges slightly higher than in the simple variant:

Definition 13 (Tiny, small, big edges)**.** *We call an edge e tiny, if $w(e) \leq 1/3$; small, if $1/3 < w(e) \leq 1/2$; and big, if $w(e) > 1/2$. We will write for the tiny, small, and big edges $T \subseteq E$, $S \subseteq E$, and $B \subseteq E$, respectively.*

At first glance one might think that the algorithm and analysis for the simple case easily extends to the general case. However, there are serious issues to resolve. We will start by giving an informal overview of the issues arising from different big edge sizes. Suppose we leave the algorithm as it is and try to get a contradiction via the dual of the configuration LP. A straight-forward choice of the edge variables would be

$$z(e) = \begin{cases} w(e) & \text{if } e \text{ is big or small and} \\ \beta w(e) & \text{if } e \text{ is tiny.} \end{cases}$$

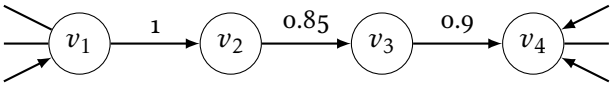


Figure 7: Path of different big edge weights

This immediately fails: Consider some path of edges that have different big sizes, see e.g. Figure 7. With $y(v_3) = z(\delta^-(v_3)) = 0.85$ the configuration consisting only of the 0.9-edge would be too large for v_3. Within such a path it seems that all big edges must have the same z-value. Also, it is easy to see that the analysis in the simple case breaks when choosing a value significantly smaller than 1. Hence, another sensible choice of edge variables would be

$$z(e) = \begin{cases} 1 & \text{if } e \text{ is big,} \\ w(e) & \text{if } e \text{ is small, and} \\ \beta w(e) & \text{if } e \text{ is tiny.} \end{cases}$$

Unfortunately, this also fails. Consider a vertex v that has two incoming big edges e_B and $e_{B'}$, e_B with weight $w(e_B) = 0.5 + \epsilon$ and $e_{B'}$ with weight $w(e_{B'}) = 1$. Suppose this vertex is prospect vertex of a small edge flip e_S of size $w(e_S) = 0.5 - \epsilon$. Then there is a configuration of value $1.5 - \epsilon$ consisting of e_B and e_S. This is too big for v, since $y(v) = z(\delta^-(v)) - 2(1 - R) = 2R$. It seems like we need some trade-off between the cases. When a situation occurs where a big edge is compatible with a small edge, we would like to fall back to the first choice of the variables (at least for this particular edge). In the upcoming proof we introduce an edge set Q that corresponds to these problematic big edges. The precise choice of the variables is highly non-trivial and in order to make it work, we also construct a more sophisticated algorithm.

As before the algorithm maintains a list P of pending flips $P_1, \dots, P_\ell$. Unlike before, however, This list contains two different types of flips. A pending flip P_k may be regular or raw. We write $P_k = (e_k^P, \text{REG})$ and $P_k = (e_k^P, \text{RAW})$, respectively. In the simple case, each edge could only appear once in P. This is slightly relaxed here. An edge e may appear once as a raw pending flip (e, RAW) and later again as a regular pending flip (e, REG). Regular pending flips behave similarly to the pending flips in the simple variant. Raw pending flips are used in some cases when we want to repel only big edges on the prospect vertex, although by the behavior of regular flips non-big edges could also be repelled. As indicated in the example above, it is sometimes very helpful, if a big edge cannot be combined with non-big edges. Indeed, when we can avoid that a vertex repels small edges, then this means we do not have to worry about such combinations.

REPELLED EDGES. The repelled edges are again defined inductively for $P_{\le 0}$, $P_{\le 1}$,... Consider the list of ℓ pending flips $P = P_{\le \ell}$ and some $k \le \ell$.

(INITIALIZATION) If $k = 0$, let every bad vertex v repel every edge $e \in \delta(v)$ w.r.t. $P_{\leq k} = P_{\leq 0}$. Furthermore, let every vertex v repel every loop from $\delta(v)$, i.e., every $e \in E$ with $\{v\} = r(e)$.

(MONOTONICITY) If $k > 0$ and v repels e w.r.t. $P_{\leq k-1}$, then let v also repel e w.r.t. $P_{\leq k}$.

The remaining rules regard $k > 0$ and e_k^P, the last pending flip in $P_{\leq k}$, and we distinguish between raw and regular pending flips.

(RAW) If $P_k = (e_k^P, \text{RAW})$, then let $s(e_k^P)$ repel all big edges in $\delta(v)$ and edges in $\delta(v)$ of weight at least $w(e_k^P)$.

From here on we will consider the case where $P_k = (e_k^P, \text{REG})$. Let $\tilde{E} = \tilde{E}(P_{\leq k-1})$ denote all edges e that are repelled by $s(e)$ w.r.t. $P_{\leq k-1}$. Furthermore, we define $E_{\geq W} = \{e \in E : w(e) \geq W\}$. Let W_0 denote the maximal $W \in (0, w(e_k^P)]$ with

$$w(\delta^-_{\tilde{E} \cup E_{\geq W}}(s(e_k^P))) + w(e_k^P) > 1 + R. \tag{14}$$

For technical reasons we define $W_0 = 0$ if no such W exists (this is an uninteresting corner case where $w(\delta^-(s(e_k^P))) + w(e_k^P) \leq 1 + R$).

(UNCRITICAL) If $P_k = (e_k^P, \text{REG})$ and $W_0 > 1/3$, then let $s(e_k^P)$ repel all edges in $\delta_{\tilde{E} \cup E_{\geq W_0}}(s(e_k^P))$.

(CRITICAL) If $P_k = (e_k^P, \text{REG})$ and $W_0 \leq 1/3$, then let $s(e_k^P)$ repel all edges in $\delta(s(e_k^P))$.

Next we will derive similar facts as we did for the simple case before.

Fact 63. *Let e_k^P be a pending flip. Then $s(e_k^P)$ repels e_k^P w.r.t. $P_{\leq k}$.*

If $P_k = (e_k^P, \text{RAW})$, this is by rule (raw). Otherwise, it follows from $W_0 \leq w(e_k^P)$.

Fact 64. *Let $e \notin P_{\leq k}$ be an edge that is not repelled by any vertex w.r.t. $P_{\leq k-1}$, and possibly by $t(e)$ (but not $s(e)$) w.r.t. $P_{\leq k}$. If the orientation of e changes and this does not affect the sets of good and bad vertices, the edges repelled by some vertex w.r.t. $P_{\leq k}$ will still be repelled after the change.*

The proof is very similar to the one in the simple case.

Proof. We first argue that the repelled edges w.r.t. $P_{\leq k'}$, $k' = 0, \ldots, k-1$ have not changed. This argument is by induction. Since the good and bad vertices do not change, the edges repelled w.r.t. $P_{\leq 0}$ do not change. Let $k' \in \{1, \ldots, k-1\}$ and assume that the edges repelled w.r.t. $k'-1$ have not changed. If $P_{k'} = (e_{k'}^P, \text{RAW})$, it is trivial that repelled edges w.r.t. $P_{\leq k'}$ have not changed. Hence, assume that $P_{k'} = (e_{k'}^P, \text{REG})$. Let W_0 be as in the definition of repelled edges w.r.t. $P_{\leq k'}$ before the change. We have to understand that e is not and was not in $\delta^-_{\tilde{E} \cup E_{\geq W_0}}(s(e_{k'}^P))$ (with $\tilde{E} = \tilde{E}(P_{\leq k'-1})$). This means that flipping it does not affect the choice of W_0 and, in particular, not the

repelled edges. Let v and v' denote the vertex e is oriented towards before the flip and after the flip, respectively. Since e was not repelled by v and v' w.r.t. $P_{\leq k'}$, it holds that $v, v' \neq s(e_k^P)$ or $w(e) < W_0$: $s(e_k^P)$ repelled all edges greater or equal W_0 in both the case (uncritial) and (critical), but e was not repelled by v or v'.

Therefore $e \notin \delta^-_{E_{\geq W_0}}(s(e_{k'}^P))$ before and after the change. Moreover, since e was not repelled by any vertex w.r.t. $P_{\leq k'-1}$ (and by induction hypothesis, it still is not), it follows that $e \notin \tilde{E}(P_{\leq k'-1})$. By this induction we have that edges repelled w.r.t. $P_{\leq k-1}$ have not changed and by the same argument as before, e is not in $\delta^-_{\tilde{E} \cup E_{\geq W_0}}(s(e_k^P))$ after the change. This means W_0 has not increased and edges repelled w.r.t. $P_{\leq k}$ are still repelled. It could be that W_0 decreases, if e was in $\delta^-_{\tilde{E} \cup E_{\geq W_0}}(s(e_k^P))$ before the flip. This would mean that the number of edges repelled by $s(e_k^P)$ increases. □

We note that W_0 (in the definition of $P_{\leq k}$) is either equal to $w(e_k^P)$ or it is the maximal value for which (14) holds. Furthermore,

Fact 65. *Let $P_k = (e_k^P, \text{REG})$ be a regular pending flip and let W_0 be as in the definition of repelled edges w.r.t. $P_{\leq k}$. If $W_0 < w(e_k^P)$, then there is an edge of weight exactly W_0 in $\delta^-(s(e_k^P))$. Furthermore, it is not a loop and it is not repelled by its other vertex, i.e., not $s(e_k^P)$, w.r.t. $P_{\leq k}$.*

Proof. We prove this for $P_{\leq k-1}$. Since the change from $P_{\leq k-1}$ to $P_{\leq k}$ only affects $s(e_k^P)$, this suffices. All edges that are repelled by their other vertex w.r.t. $P_{\leq k-1}$ (in particular, loops) are in $\tilde{E}$. Recall that $w(\delta^-_{\tilde{E} \cup E_{\geq W_0}}(s(e_k^P)) + w(e_k^P) > 1 + R$. Assume toward contradiction there is no edge of weight W_0 in $\delta^-(s(e_k^P))$, which is not in $\tilde{E}$. This means there is some $\epsilon > 0$ such that $\delta^-_{\tilde{E} \cup E_{\geq W_0 + \epsilon}}(s(e_k^P)) = \delta^-_{\tilde{E} \cup E_{\geq W_0}}(s(e_k^P))$. Hence, W_0 is not maximal. □

7.3.1 *Algorithm*

The general structure of the algorithm (see Alg. 7) is the same as in the simplified case. The difference are the conditions on when an regular pending flips can be added to P and the role of the set Q. Raw pending flips are added to P by the same rule as before: They must be repelled by the vertex they are currently oriented towards and must not be repelled by the other. The interesting novelty is when to add regular pending flips. Tiny edges will be added as soon as possible and there is no further condition. Small edges are added when either there is only one (or no) big edge oriented towards the prospect vertex or when the small edge fits into a configuration with one of the big edges on the vertex. Finally, big edges are added as regular flips only when there are less than 2 big edges on the prospect vertex. Moreover, either the edge must be in Q or none of the following holds:

1. The big edge on the prospect vertex is bigger than R;
2. The big edge on the prospect vertex is not in $\tilde{E}$ (which means it can be added as a pending flip later);

Input: Weighted multigraph $G = (V, E, r, w)$ with $\mathrm{OPT}^* = 1$
Result: Orientation $s, t : E \to V$ with maximum weighted in-degree $1 + R$
let $s, t : E \to V$ map source and target vertices to each edge, i.e., $\{s(e), t(e)\} = r(e)$ for all $e \in E$, such that there are at most two big edges oriented towards each vertex ;
$\ell \leftarrow 0$; // number of pending edges P to flip
while *there exists a bad vertex* **do**
  if *there exists a valid regular flip* $(e, \textsc{reg}) \in P$ **then**
    let k be minimal such that e is repelled by $t(e)$ w.r.t. $P_{\leq k}$;
    exchange $s(e)$ and $t(e)$;
    $P \leftarrow P_{\leq k}$; $\ell \leftarrow k$; // Forget pending flips $P_{k+1}, \ldots, P_\ell$
    $Q_\ell \leftarrow \emptyset$;
  else
    for $e \in E$ *in non-decreasing order of* $w(e)$ **do**
      if `IsRawAddable`$(e, P_{\leq \ell}, Q_{\leq \ell})$ **then**
        $P_{\ell+1} \leftarrow (e, \textsc{raw})$; $Q_{\ell+1} \leftarrow \emptyset$; $\ell \leftarrow \ell + 1$; // Append e to P
        break;
      else if `IsRegularAddable`$(e, P_{\leq \ell}, Q_{\leq \ell})$ **then**
        $P_{\ell+1} \leftarrow (e, \textsc{reg})$; $Q_{\ell+1} \leftarrow \emptyset$; $\ell \leftarrow \ell + 1$; // Append e to P
        break;
  loop
    if *there exists an* $(e, \textsc{raw}) \in P$ *with* $e \notin Q_{\leq \ell}$ *and* $1/2 < w(e) \leq 0.6$ *such that* $t(e)$ *repels an outgoing, non-loop edge* e' *with* $w(e) + w(e') \leq 1$ **then**
      // e is not too big and interferes with other edges
      $Q_\ell \leftarrow Q_\ell \cup \{e\}$;
    else
      break;

Algorithm 7: Local search algorithm for general Graph Balancing

Function `IsRawAddable`(e,P,Q)
return
$$\begin{cases} \text{True} & \text{if } e \text{ is repelled by } t(e) \text{ and not repelled by } s(e) \text{ w.r.t. } P, \\ \text{False} & \text{otherwise ;} \end{cases}$$
Function `IsRegularAddable`(e,P,Q)
if $(e, \textit{REG}) \in P$ *or* $(e, \textit{RAW}) \notin P$ **then return** False;
if *e is tiny* **then**
return True ;
else if *e is small* **then**
$$\textbf{return} \begin{cases} \text{True} & \text{if } |\delta_B^-(s(e))| \leq 1, \\ \text{True} & \text{if } w(e) + w(e_B) \leq 1 \text{ for some } e_B \in \delta_B^-(s(e)), \\ \text{False} & \text{otherwise ;} \end{cases}$$
else if *e is big* **then**
Let F denote all big edges $e \in P$ such that there is no smaller flip $e' \in P$ towards $s(e)$, i.e., $w(e') < w(e)$ and $s(e') = s(e)$;
// Maintain invariant of $|\delta_B^-(s(e))| \leq 2$
if $|\delta_B^-(s(e))| = 2$ **then return** False;
// From here on $|\delta_B^-(s(e))| \leq 1$
return
$$\begin{cases} \text{True} & \text{if } e \in Q, \\ \text{True} & \text{if } w(\delta_B^-(s(e)) \leq R,\ \delta_B^-(s(e)) \subseteq \tilde{E}, \text{ and } \delta_F^-(s(e)) \subseteq Q, \\ \text{False} & \text{otherwise ;} \end{cases}$$

3. The big edge on the prospect vertex is not in Q and is the smallest pending flip towards its other vertex.

For intuition, these 3 cases should identify paths of big edges as described in the beginning of the section, i.e., these edges will each get a z-value of 1. Inside such paths we would like the big edges to be only raw pending flips. Case 2 is only a temporary state. It is not clear, yet, how the big edge on the prospect vertex behaves. Case 1 says that when the next edge is bigger than R, then this should be a path of big edges and we do not want to add the regular pending flip. Case 3 deals with big edges smaller than R. This has to do with the edge set Q. Let us discuss the idea behind Q. Q is a subset of big raw pending flips. These edges are identified as big edges that should not be part of such a path of big edges. In particular, edges in Q may be added as a regular flip and for the condition of adding other big edges as regular pending flips, we ignore edges in Q. Edges are added to Q when a situation occurs where a non-big edge is compatible with this edge.

Note that when raw pending flips are added again as regular pending flips we no longer require that the edge is not repelled by the prospect vertex. It is sufficient that this was true when it was previously added as a raw pending flip. By Fact 63 this would not hold anyway.

7.3.2 *Framework for analysis*

The analysis shares the same basic structure with the simple case.

Lemma 66. *The algorithm terminates after finitely many iterations of the main loop.*

Lemma 67. *If there at least one bad vertex remaining, then there is either a valid regular flip in P or a flip that can be added to P.*

We also have to show that there exists an initial orientation of edges such that every vertex gets at most two big edges. In fact, we could also easily do it with one edge, but this invariant appears to be hard to maintain during a local search. For at most two edges, take the solution x of the configuration LP and orient every edge e towards the vertex in $r(e) = \{u, v\}$ that the solution prefers, i.e., towards u if $\sum_{C \in \mathcal{C}(u,1): e \in C} x_{u,C} \geq \sum_{C \in \mathcal{C}(v,1): e \in C} x_{v,C}$ and towards v otherwise. This is already a 2-approximation, because

$$\sum_{C \in \mathcal{C}(u,1): e \in C} x_{u,C} \geq 0.5,$$

if u gets edge e. Every vertex v has $\sum_{C \in \mathcal{C}(v,1)} x_{v,C} \leq 1$ and a configuration can only contain one big edge. Therefore no more than 2 edges can be oriented towards each vertex.

Fact 68. *At every time during the execution of the algorithm $|\delta_B^-(v)| \leq 2$ for all $v \in V$.*

Proof of Lemma 66. Let ℓ be the length of P. We define a potential function $s(P)$ as

$$s(P) = (g, |\tilde{E}(P_{\leq 0})|, |\tilde{E}(P_{\leq 1})|, \ldots, |\tilde{E}(P_{\leq \ell})|, -1),$$

where g denotes the number of good vertices. Like in the simple case, we will argue that this vector increases lexicographically after every iteration of the main loop. Since the number of possible vectors is finite, so is the running time. If a pending flip is added, the increase is obvious. Now consider the case where a pending flip $e \in P$ is executed. The number of good vertices does not decrease, since the algorithm only executes valid flips. If the number of good vertices increases, we are done. Hence, assume otherwise. Let v be the vertex e was oriented to before the flip and v' the vertex afterwards. We denote by ℓ the length of P before the flip was executed and by ℓ' the length afterwards. Recall that ℓ' was chosen such that e is repelled by v w.r.t. $P_{\leq \ell'}$, but not w.r.t. $P_{\leq \ell'-1}$. Let P_k be the raw pending flip for e. Then $k \geq \ell' + 1$, since otherwise ℓ' would not be minimal. Furthermore, e was not repelled by v' w.r.t. $P_{\leq k-1}$ (in particular, not w.r.t. $P_{\leq \ell'}$) when P_k was added. Indeed, e is still not repelled by v' w.r.t. $P_{\leq \ell'}$, since any change to the repelled edges would trigger the removal of all pending flips $P_{\ell'+1}, P_{\ell'+2}, \ldots$ However, P_k is still in the list.

We conclude that the premise of Fact 64 is fulfilled and therefore edges that were repelled w.r.t. $P_{\leq k}$ (before the flip of e) for any $k \leq \ell'$ are still and, in particular, $|\tilde{E}(P_{\leq k})|$ has not decreased. Finally, $|\tilde{E}(P_{\leq \ell'})|$ has increased: e itself was not in that set, since it is not repelled by v'. After the flip we have $s(e) = v$ and since e is repelled by v, it is now in this set. □

Proof of Lemma 67. We assume the contrary and show that $\mathrm{OPT}^* > 1$. This proof is by demonstrating the dual of the configuration LP is unbounded. For this purpose, in the following we will define a solution z, y for the dual. Throughout the proof we will use the parameter $\beta = 1.03$. As before let $\tilde{E} \subseteq E$ denote all edges e that are repelled by $s(e)$. This includes all edges that are repelled by $t(e)$, since otherwise e could be added to P or it is in P and then $e \in \tilde{E}$ follows from Fact 63. We let Q be the edges in $Q_{\leq \ell}$ at the time the algorithm gets stuck. Furthermore, define F as the set of big edges $e \in P$ such that there is no $e' \in \delta_P^+(s(e))$ with $w(e') < w(e)$. Let H denote all edges with weight greater than R which are not in F. For clarity a glossary of important symbols is provided in Table 1. The relations between the edge sets are illustrated in Figure 8.

We define critical vertex as in the simple case: Observe that by definition of repelled edges, a vertex v repels either

- only big edges; or
- edges with weight at least a certain threshold $W > 1/3$ and edges from $\delta^-_{\tilde{E}}(v)$; or
- all edges.

V	All vertices.
E	All edges.
$\tilde{E}$	Edges e that are repelled by $s(e)$ (in particular, those that are repelled by $t(e)$).
P	Pending flips.
T	Tiny edges, i.e., all edges $e \in E$ with $w(e) \leq 1/3$.
S	Small edges, i.e., all edges $e \in E$ with $1/3 < w(e) \leq 1/2$.
B	Big edges, i.e., all edges $e \in E$ with $1/2 < w(e)$.
F	Big edges in P such that there is no smaller pending flip towards the same vertex.
Q	Big edges $e \in P$ with $0.5 < w(e) \leq 0.6$ such that $t(e)$ repels a non-loop edge e' with $w(e) + w(e') \leq 1$.
H	Very big edges $e \in E \setminus F$ (with $w(e) > R$).

Table 1: Glossary of important symbols

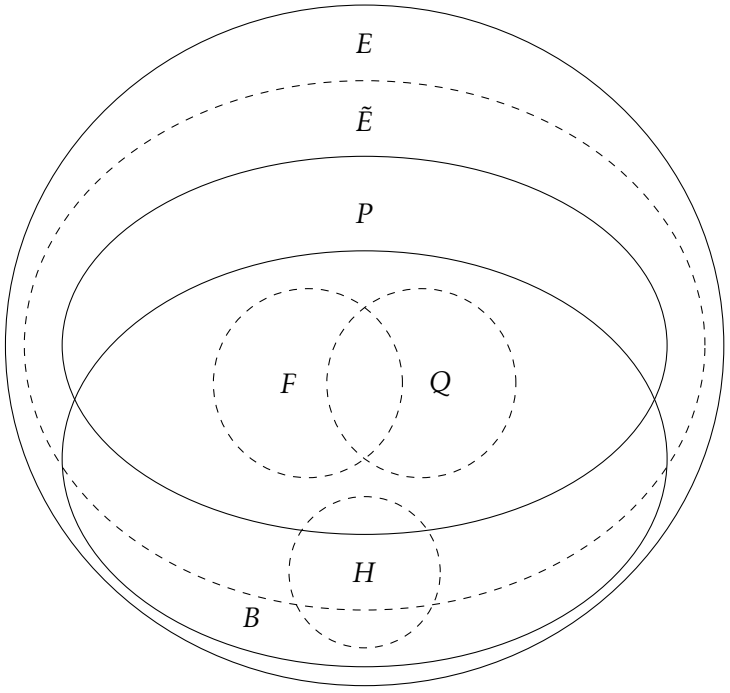


Figure 8: Inclusion relations of edge sets (omitting S and T). Dashed lines are only for readability

We call vertices of the last kind *critical*. For every $e \in E \setminus \tilde{E}$, set $z(e) = 0$. For every $e \in \tilde{E}$, set

$$z(e) = \begin{cases} 1 & \text{if } 1/2 < w(e) \text{ and } e \in F \setminus Q, \\ \min\{w(e), R\} & \text{if } 1/2 < w(e) \text{ and } e \notin F \setminus Q, \\ w(e) & \text{if } 1/3 < w(e) \leq 1/2, \text{ and} \\ \beta w(e) & \text{if } w(e) \leq 1/3. \end{cases}$$

On average, we want to set each $y(v)$ to $z(\delta^-(v)) = z(\delta^-_{\tilde{E}}(v))$. However, there are two kinds of amortization between vertices.

CRITICAL AMORTIZATION. If v is good and critical, but is not a prospect vertex of a tiny flip, set $a_v := \beta - 1$. If v is is good and prospect vertex of tiny flip (in particular, critical), set $a_v := -(\beta - 1)$. Otherwise, set $a_v = 0$.

BIG AMORTIZATION. Set

$$b_v := (|\delta^+_F(v)| - |\delta^-_F(v)|) \cdot (1 - R) - \sum_{e \in \delta^+_{F \cap Q}(v)} [R - w(e)] + \sum_{e \in \delta^-_{F \cap Q}(v)} [R - w(e)].$$

Finally, define $y(v) = z(\delta^-(v)) + a_v + b_v$ if v is a good vertex and $y(v) = z(\delta^-(v)) + a_v + b_v - \mu$ otherwise. Here $\mu = 0.01$ is some sufficiently small constant. By the almost identical argument as in the simple case (which we will repeat for completeness) we get the following.

Claim 69. $\sum_{e \in E} z(e) > \sum_{v \in V} y(v)$.

It remains to show that for each $v \in V$ and $C \in \mathcal{C}(v, 1)$ it holds that $y(v) \geq z(C)$. This proof is much more complicated than in the simple case and for that purpose it is divided into two cases.

Claim 70. *Let $v \in V$ with $|\delta^-_B(v)| \leq 1$. Let $C \in \mathcal{C}(v, 1)$. Then $y(v) \geq z(C)$.*

Claim 71. *Let $v \in V$ with $|\delta^-_B(v)| = 2$. Let $C \in \mathcal{C}(v, 1)$. Then $y(v) \geq z(C)$.*

Together, these claims and Corollary 68 fulfill the premise of Lemma 52 and therefore imply that the configuration LP is not feasible for $\mathrm{OPT}^* = 1$, a contradiction. □

Proof of Claim 69. First we note that

$$\sum_{v\in V} b_v = (1-R)\underbrace{\Big(\sum_{v\in V}|\delta_F^+(v)| - \sum_{v\in V}|\delta_F^-(v)|\Big)}_{=0} + R\cdot\underbrace{\Big(\sum_{v\in V}|\delta^-_{F\cap Q}(v)| - \sum_{v\in V}|\delta^+_{F\cap Q}(v)|\Big)}_{=0} + \underbrace{\sum_{v\in V} w(\delta^+_{F\cap Q}(v)) - \sum_{v\in V} w(\delta^-_{F\cap Q}(v)|)}_{=0} = 0$$

Moreover, we have that $\sum_{v\in V} a_v \leq 0$: This is because at least half of all good vertices that are critical are prospect vertices of tiny flips. When a vertex v is critical but not prospect vertex of a tiny flip, there must be a regular non-tiny flip (e_k^P, REG) with $s(e_k^P) = v$ such that in the definition of repelled edges for $P_{\leq k}$ we have $W_0 \leq 1/3$. By Fact 65 there exists an edge of weight W_0 (hence, tiny) in $\delta^-(v)$ which is not repelled by its other vertex $v' \neq v$ w.r.t. $P_{\leq k}$. We argue later that this edge has already been in $\delta^-(v)$ when the list of pending flips has consisted only of the k flips in $P_{\leq k}$. Since a tiny flip will always be added before the next non-tiny regular flip (edges of minimal weight are chosen), we know that the next pending flip e_{k+1}^P must be a tiny flip and $s(e_{k+1}^P)$ is the prospect vertex of a tiny flip. Also, this will become prospect vertex of a regular tiny flip before the next non-tiny pending flip can be added. Note that no vertex can be prospect vertex of two regular tiny flips and therefore the ratio holds.

The reason why the edge has already been in $\delta^-(s(e_k^P))$ is the following. When an edge e is flipped towards $v = s(e_k^P)$ and it is repelled by v w.r.t. $P_{\leq k}$, then e must have been a flip in P earlier in the list than e_k^P. This means that at least $e_k^P, \ldots, e_\ell^P$ are removed from P. Because e_k^P is still in P, we can therefore assume that this has not happened. If W_0 in the definition of the repelled edges w.r.t. $P_{\leq k}$ has not decreased since the last time the list consisted only of $P_{\leq k}$, we are done, since this would mean the edge of weight W_0 has been repelled by v all this time and could not have been added to $\delta^-(v)$. This is indeed the case. W_0 can only decrease when an edge from $\delta^-_{\tilde{E}\cup E_{\geq W_0}}(v)$ is flipped. This, on the other hand, causes at least $e_{k+1}^P, \ldots, e_\ell^P$ to be removed from P. This finished the proof of $\sum_{v\in V} a_v \leq 0$. We conclude,

$$\sum_{e\in E} z(e) \geq \sum_{v\in V}\sum_{e\in\delta^-(v)} z(e) + \underbrace{\sum_{v\in V}[b_v + a_v]}_{\leq 0} > \sum_{v\in V} y(v),$$

where the strict inequality holds because of the definition of $y(v)$ and because there exists at least one bad vertex. □

Before we prove the other two claims, let us recall a fact from the simple version that still holds in a slightly modified variant.

Fact 72. *Let $(e_k^P, \textsc{reg})$ be a regular pending flip. Then*

$$w(\delta^-_{\tilde{E}}(s(e_k^P))) + w(e_k^P) > 1 + R.$$

This fact follows directly from the definition of repelled edges and because there is no valid flip. The next three facts are related to the set Q or regular pending flips. They state invariants that one would also intuitively expect, since they correspond to the requirements for adding edges to Q or as regular flips.

Fact 73. *Let $(e_k^P, \textsc{raw})$ be a big raw pending flip ($w(e_k^P) > 0.5$). Then $e_k^P \in Q$, if and only if $w(e_k^P) \leq 0.6$ and there is a non-loop edge $e' \in \delta^+_{\tilde{E}}(t(e_k^P))$ with $w(e_k^P) + w(e') \leq 1$.*

Proof. If $w(e_k^P) \leq 0.6$ and there is a non-loop edge $e' \in \delta^+_{\tilde{E}}(t(e_k^P))$ with $w(e_k^P) + w(e') \leq 1$, but $e_k^P \notin Q$, then by definition of the algorithm e_k^P would have been added to Q in the last iteration of the main loop. A contradiction.

Suppose that $e_k^P \in Q_{k'}$. When it was added to $Q_{k'}$ there was such a non-loop edge $e' \in \delta^+(t(e_k^P))$ that was repelled by $t(e_k^P)$ w.r.t. $P_{\leq k'}$. If this edge was flipped (and therefore removed from $\delta^+(t(e_k^P))$, then $Q_{k'}$ would have been deleted, which did not happen. What is left to show is that e' is still repelled by $s(e_k^P)$. If e' is not repelled by $s(e_k^P)$ w.r.t. $P_{\leq k'}$ anymore, there would have to be some edge e'' that was removed from $\delta^-(s(e_k^P))$ and which was also repelled by $s(e_k^P)$ w.r.t. $P_{\leq k'}$. Then, however, $Q_{k'}$ would have been removed. □

Fact 74. *Let $e \in E$ be big, i.e., $w(e) > 0.5$, and $P_k = (e, \textsc{raw})$ a raw pending flip for e. Then $(e, \textsc{reg}) \in P$, if and only if $|\delta^-_B(s(e))| \leq 1$ and (a) $e \in Q$, or (b) $w(\delta^-_B(s(e))) \leq R$ and $\delta^-_F(s(e)) \subseteq Q$.*

Proof. If $|\delta^-_B(v)| \leq 1$ and (a), but $(e, \textsc{reg}) \notin P$, then this is a regular flip that can be added to P, which contradicts the assumption that the algorithm is stuck. A small twist has to be considered, if $|\delta^-_B(v)| \leq 1$ and (b), but $(e, \textsc{reg}) \notin P$: Then the flip can only be added, if $\delta^-_B(s(e)) \subseteq \tilde{E}$. Fortunately, this is the case. Because of the raw pending flip, all edges in $\delta^-_B(s(e))$ are repelled by $s(e)$ and $\tilde{E}$ contains all edges e' that are repelled by $t(e')$ (not only those repelled by $s(e')$, as elaborated earlier). Note that during execution of the algorithm (when it is not stuck) this is not a tautology.

Now suppose that the pending flip $P_{k'} = (e, \textsc{reg})$ exists ($k' > k$). We will check that the conditions hold. We do know that $|\delta^-_B(v)| \leq 1$ and (a) or (b) at the time $P_{k'}$ was added. First, consider the case where (a) was true. Since $Q_{\leq k'-1} \subseteq Q_{\leq \ell} = Q$, we have (a) is still true. Since $P_{k'}$ was added, no big edges can have been flipped towards $s(e)$, because they are repelled by $s(e)$. Therefore, $|\delta^-_B(v)| \leq 1$ still holds. Next, assume (b) was true when $P_{k'}$ was added. As before, $|\delta^-_B(v)| \leq 1$ holds. For the same reason, $w(\delta^-_B(v)) \leq R$ must still be true. Finally, consider the condition $\delta^-_F(s(e)) \subseteq Q$. Assume that $\delta^-_F(s(e)) \neq \emptyset$ and let $\{e_F\} = \delta^-_F(s(e))$. This edge has already been in $\delta^-(s(e))$ when $P_{k'}$ was added (see above). It also must have been in P already, because otherwise $e' \in B \setminus \tilde{E}$, and $P_{k'}$ cannot be added when such an edge

exists. Furthermore, $e' \in F$ has been true, since if it is not in F, then there would have been a smaller flip towards $s(e')$ and this would remain true. e', however, is in F and thus, it has been in F when $P_{k'}$ was added. By condition of adding $P_{k'}$ this means e' has been in Q and therefore still is. We conclude, $\delta_F^-(s(e)) = \{e'\} \subseteq Q$. □

Fact 75. *Let $e \in E$ be small, i.e., $1/3 < w(e) \leq 1/2$. Let $P_k = (e, \text{RAW})$ be a raw pending flip. Then $(e, \text{REG}) \in P$, if and only if $|\delta_B^-(s(e))| \leq 1$ or there exists $e_B \in \delta_B^-(s(e))$ with $w(e) + w(e_B) \leq 1$.*

Proof. Clearly, if $|\delta_B^-(s(e))| \leq 1$ or there is some $e_B \in \delta_B^-(s(e))$ with $w(e) + w(e_B) \leq 1$, but $(e, \text{REG}) \notin P$, then this pending flip could be added, which contradicts the assumption that the algorithm is stuck.

Now suppose that $P_{k'} = (e, \text{REG})$ is a regular pending flip for e. As long as $P_{k'}$ remains in P, no big edge can be flipped towards $s(e)$, since they are all repelled by $s(e)$. If $|\delta_B^-(s(e))| \leq 1$ at the time $P_{k'}$ was added, then this is still true. Otherwise, there was a big edge $e_B \in \delta_B^-(s(e))$ with $w(e) + w(e_B) \leq 1$ at that time. If it is still in $\delta^-(s(e))$, then we are done. If, on the other hand, it was flipped then at most one big edge remains that is oriented towards $s(e)$, i.e., $|\delta_B^-(s(e))| \leq 1$. □

7.3.3 *Proof of Claim 70*

Recall, this claim is the inequality $z(C) \leq y(v)$ for the case of $|\delta_B^-(v)| \leq 1$.

Case 1: v is a bad vertex.

Note that no pending flip can go towards a bad vertex, hence $\delta_F^+(v) = \emptyset$. This implies that if C contains any edge from F, it must be in $\delta^-(v)$. Therefore, $z(C) \leq z(\delta_F^-(v)) - w(\delta_F^-(v)) + \beta$. Thus,

$$\begin{aligned} y(v) &= z(\delta^-(v)) + b_v - \mu \\ &\geq z(\delta_F^-(v)) - w(\delta_F^-(v)) + w(\delta_{\tilde{E}}^-(v)) - (1 - R) - \mu \\ &> z(\delta_F^-(v)) - w(\delta_F^-(v)) + 1 + R - 1 + R - \mu \\ &\geq z(\delta_F^-(v)) - w(\delta_F^-(v)) + \beta \geq z(C). \end{aligned}$$

For the first inequality we distinguish between $|\delta_H^-(v)| = 0$ and $|\delta_H^-(v)| = 1$. In the former, we have that $b_v \geq -(1 - R)$ and $z(e) \geq w(e)$ for all $e \in \delta_{\tilde{E}}^-(v)$. In the latter, we have that $b_v = 0$ and $z(\delta_H^-(v)) \geq w(\delta_H^-(v)) - (1 - R)$ (and $z(e) \geq w(e)$ for all $e \in \delta_{\tilde{E} \setminus H}^-(v)$). For the second to last inequality, we use that $w(\delta_{\tilde{E}}^-(v)) = w(\delta^-(v)) > 1 + R$ by the definition of a bad vertex.

Case 2: v is good and prospect vertex of a tiny flip.

Since v is prospect vertex of a smaller flip than any big edge, $\delta_F^+(v) = \emptyset$ and therefore again $z(C) \leq z(\delta_F^-(v)) - w(\delta_F^-(v)) + \beta$. Moreover, since v is

prospect vertex of a tiny flip, i.e., of an edge of weight at most $1/3$, it must be that $w(\delta^-_{\tilde{E}}(v)) > 1 + R - 1/3 = R + 2/3$. Thus,

$$\begin{aligned} y(v) &= z(\delta^-(v)) + b_v - (\beta - 1) \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + w(\delta^-_{\tilde{E}}(v)) - (1 - R) - (\beta - 1) \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 2R + 2/3 - \beta \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + \beta \geq z(C). \end{aligned}$$

For the first inequality, we argue in the same way as in the previous case.

Case 3: v is good, not prospect vertex of a tiny flip, but prospect vertex of a small flip.

Like in the last case we have $\delta^+_F(v) = \emptyset$. Let e_S be the smallest edge with a pending flip towards v. If $|\delta^-_Q(v)| = 1$, then

$$\begin{aligned} y(v) &\geq w(\delta^-_{\tilde{E}}(v)) - \underbrace{(1 - 2R + w(\delta^-_Q(v)))}_{=b_v} \\ &> 1 + R - w(e_S) - (1.6 - 2R) \geq 3R - 1.1 \geq \beta \geq z(C). \end{aligned}$$

We can therefore assume that $|\delta^-_Q(v)| = 0$. If in addition $|\delta^-_{(F \setminus Q) \cup H}(v)| = 0$ holds, then

$$y(v) \geq z(\delta^-(v)) \geq 1 + R - w(e_S) \geq 1 + R - 0.5 \geq \beta \geq z(C).$$

In the remainder of Case 3, assume that $|\delta^-_{(F \setminus Q) \cup H}(v)| = 1$. In particular, if there is an edge from F in C or $\delta^-(v)$, then it is not in Q and therefore has a z-value of 1. Moreover, $z(\delta^-_B(v)) + b_v = R$ by a simple case distinction.

CASE 3.1: v IS CRITICAL. If $C \cap F \neq \emptyset$, then

$$\begin{aligned} y(v) &\geq z(\delta^-(v)) + b_v + (\beta - 1) \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 1 + R - w(e_S) - (1 - R) + (\beta - 1) \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 2R - 1.5 + \beta \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 0.5 + 0.5\beta \geq z(C). \end{aligned}$$

If $C \cap F = \emptyset$, and $\delta^-_{\tilde{E}}(v)$ contains a small edge,

$$\begin{aligned} y(v) &= z(\delta^-(v)) + b_v + (\beta - 1) \\ &\geq \underbrace{z(\delta^-_B(v)) + b_v}_{=R} + \underbrace{z(\delta^-_{\tilde{E} \setminus B}(v))}_{>1/3>1-R} + (\beta - 1) \geq \beta \geq z(C). \end{aligned}$$

If $C \cap F = \emptyset$, and $\delta^-_{\tilde{E}}(v)$ contains no small edge,

$$\begin{aligned} y(v) = z(\delta^-(v)) + b_v + (\beta - 1) &\geq z(\delta^-_B(v)) + b_v + z(\delta^-_{\tilde{E} \setminus B}(v)) + (\beta - 1) \\ &\geq R + \beta(R - 0.5) + \beta - 1 \geq \beta \geq z(C). \end{aligned}$$

CASE 3.2: v IS UNCRITICAL AND $\delta^-_{\tilde{E}}(v)$ CONTAINS A SMALL EDGE. Note that $C \cap T \cap \tilde{E} \subseteq \delta^-(v)$ and, in particular, $z(C \cap T) \leq z(\delta^-_T(v))$, since otherwise there would be a tiny flip that can be added to P. If C contains no small edge from $\tilde{E}$ or $C \cap F = \emptyset$, then

$$y(v) \geq z(\delta^-_B(v)) + z(\delta^-_T(v)) + 1/3 + b_v \geq 1 + z(\delta^-_T(v)) \geq z(C).$$

Now assume that C contains $\delta^-_F(v) \neq \emptyset$ and a small edge $e \in \tilde{E}$. If $\delta^-_{\tilde{E}}(v)$ contains exactly one small edge e' and $w(e') \leq w(e)$,

$$\begin{aligned} y(v) &\geq z(\delta^-_F(v)) + \beta(w(\delta^-_{\tilde{E}}(v)) - w(e') - w(\delta^-_F(v))) \\ &\quad + w(e') - (1-R) \\ &\geq 1 + \beta(1 + R - 0.5 - w(e') - w(\delta^-_F(v))) + w(e') - \beta(R - 0.5) \\ &= 1 + \beta(1 - w(e') - w(\delta^-_F(v))) + w(e') \\ &\geq 1 + \beta(1 - w(e) - w(\delta^-_F(v))) + w(e) \geq z(C). \end{aligned}$$

If $\delta^-_{\tilde{E}}(v)$ contains exactly one small edge e' and $w(e') > w(e)$, then since e is repelled, it must that $w(e) \geq w(e_S)$. Thus,

$$\begin{aligned} y(v) &\geq z(\delta^-_F(v)) + \beta(w(\delta^-_{\tilde{E}}(v)) - w(e') - w(\delta^-_F(v))) \\ &\quad + w(e') - (1-R) \\ &\geq 1 + \beta(1 + R - w(e_S) - w(e') - w(\delta^-_F(v))) \\ &\quad + w(e') - \beta(R - 0.5) \\ &= 1 + \beta(1.5 - w(e_S) - w(e') - w(\delta^-_F(v))) + w(e') \\ &\geq 1 + \beta(1 - w(e_S) - w(\delta^-_F(v))) + 0.5 \\ &\geq 1 + \beta(1 - w(e) - w(\delta^-_F(v))) + w(e) \geq z(C). \end{aligned}$$

If $\delta^-_{\tilde{E}}(v)$ contains at least two small edges and one of them is e,

$$y(v) \geq 1 + z(\delta^-_T(v)) + w(e) + 1/3 - (1-R) \geq 1 + w(e) + z(\delta^-_T(v)) \geq z(C).$$

If $\delta^-(v)$ contains at least two small edges and none of them is e, then $e \in \delta^+_{\tilde{E}}(v)$. Furthermore, $w(e) + w(\delta^-_F(v)) \leq 1$. If $w(\delta^-_F(v)) \leq 0.6$, then the edge would be in Q, which is not the case. This implies that $w(\delta^-_F(v)) > 0.6$ and therefore $w(e) < 0.4$. Thus,

$$y(v) \geq 1 + z(\delta^-_T(v)) + 2/3 - (1-R) \geq 1 + 0.4 + z(\delta^-_T(v)) \geq z(C).$$

CASE 3.3: v IS NOT CRITICAL AND $\delta^-_{\tilde{E}}(v)$ CONTAINS NO SMALL EDGE. If $C \cap F \neq \emptyset$, then

$$\begin{aligned} y(v) &\geq z(\delta^-(v)) - (1-R) \\ &\geq z(\delta^-_F(v)) + \beta(1 + R - 0.5 - w(\delta^-_F(v)) - (1-R) \\ &\geq 1 + \beta(1 - w(\delta^-_F(v)) \geq z(C). \end{aligned}$$

We now assume that C contains no edge from F. If C contains exactly one edge that is small or big (not from F),

$$y(v) \geq z(\delta^-_B(v)) + z(\delta^-_T(v)) + b_v = R + z(\delta^-_T(v)) \geq z(C).$$

If C contains a big edge $e_B \notin F$ and a small edge $e \in \tilde{E}$, then $w(e) \geq w(e_S)$, since otherwise e would not be repelled and can be added to P. Thus,

$$\begin{aligned} y(v) &\geq z(\delta_B^-(v)) + z(\delta_T^-(v)) + b_v \\ &\geq R + \beta(1 + R - w(\delta_B^-(v)) - w(e_S) \\ &\geq R + \beta(R - w(e)) = R + \beta(0.5 - w(e)) + \underbrace{\beta(R - 0.5)}_{\geq (1-R)} \\ &\geq 1 + \beta(1 - w(e_B) - w(e)) \geq z(C). \end{aligned}$$

What is left to resolve is when $C \cap \tilde{E}$ contains 2 small edges (in particular, no big edges). Then these edges must be of size at least $w(e_S)$ (again, they could be added to P, otherwise). Therefore,

$$\begin{aligned} y(v) \geq z(\delta_B^-(v)) + z(\delta_{\tilde{E}\setminus B}^-(v)) + b_v &\geq R + \beta(w(\delta_{\tilde{E}}^-(v)) - w(\delta_B^-(v))) \\ &\geq R + \beta(w(\delta_{\tilde{E}}^-(v)) - 1) \geq R + \beta(R - w(e_S)). \end{aligned}$$

It follows that

$$\begin{aligned} z(C) &\leq 2w(e_S) + \beta(1 - 2w(e_S)) \\ &\leq y(v) + 2w(e_S) - R + \beta(1 - R - w(e_S)) \\ &\leq y(v) + 2 \cdot 0.5 - R + \beta(1 - R - 0.5) \\ &= y(v) + 1 - R + \beta(0.5 - R) \leq y(v). \end{aligned}$$

Case 4: v is good and not prospect vertex of a small/tiny flip.

Since there can be only one big pending flip towards v and there is no small or tiny flip, we have that $\delta_F^+(v) = \delta_P^+(v)$. If $|\delta_F^+(v)| = |\delta_P^+(v)| = 0$, then v does not repel any edges. In particular, $\delta_F^-(v) = \emptyset$ and therefore $b_v = 0$. Moreover, every $e \in \delta(v)$ with $z(e) > 0$ must be in $\delta^-(v)$. This implies $z(C) \leq z(\delta^-(v)) \leq y(v)$. We assume in the remainder that $|\delta_F^+(v)| = 1$ (note that $|\delta_F^+(v)| \leq 1$ always holds).

CASE 4.1: $|\delta_Q^+(v)| = 0$. If $|\delta_{(F\setminus Q)\cup H}^-(v)| = 1$, then by Fact 74 there is no regular pending flip towards v Hence, v repels only big edges and

$$\begin{aligned} y(v) &\geq z(\delta_F^-(v)) + z(\delta_{\tilde{E}\setminus F}^-(v)) + b_v \\ &\geq \begin{cases} 1 + z(\delta_{\tilde{E}\setminus B}^-(v)) + 0 \geq z(C) & \text{if } |\delta_{F\setminus Q}^-(v)| = 1, \\ R + z(\delta_{\tilde{E}\setminus B}^-(v)) + (1 - R) \geq z(C) & \text{if } |\delta_H^-(v)| = 1. \end{cases} \end{aligned}$$

Assume now that $\delta_{(F\setminus Q)\cup H}^-(v) = \emptyset$. We will argue that also $\delta_{Q\cap F}^-(v) = \emptyset$: Suppose toward contradiction there is an edge $e_Q \in \delta_{F\cap Q}^-(v)$. Let k be the index at which e_Q was added to Q, that is to say, $e_Q \in Q_k$. Since then $P_{\leq k}$ has not changed or else Q_k would have been deleted. Also, e_Q was already in $\delta^-(v)$, because e_Q is a flip in $P_{\leq k}$ and if the orientation had changed, Q_k again would have been deleted. Because e_Q was added to Q_k, there was an

outgoing non-loop edge $e' \in \delta(v)$ with $w(e') + w(e_Q) \leq 1$ that was repelled by v. Since the edge $\delta^+_{F\setminus Q}(v)$ is the only pending flip towards v, it must be a regular pending flip $P_{k'} = (e_F, \text{REG})$ with $k' \leq k$. However, at the point of time when $P_{k'}$ was added, e_Q was not in Q, yet, so it was either not yet in P and therefore in $B \setminus \tilde{E}$ or it was in P and therefore also in F. Either way, the regular flip $P_{k'}$ would not have been added, since $\delta^-_B(v) \not\subseteq \tilde{E}$ or $\delta^-_F(v) \not\subseteq Q$ at that time. A contradiction.

Together we get $|\delta^-_{F\cup H}(v)| = 0$ and $b_v = 1 - R$. Furthermore, by Fact 74 it must be that $\delta^+_F(v)$ is a regular pending flip towards v. If v is not critical,

$$\begin{aligned} y(v) &\geq z(\delta^-(v)) - w(\delta^-_{\tilde{E}}(v)) + w(\delta^-_{\tilde{E}}(v)) + (1-R) \\ &\geq z(\delta^-(v)) - w(\delta^-_{\tilde{E}}(v)) + 1 + R - w(\delta^+_F(v)) + (1-R) \\ &= z(\delta^-(v)) - w(\delta^-_{\tilde{E}}(v)) + z(\delta^+_F(v)) + (1 - w(\delta^+_F(v))) \geq z(C). \end{aligned}$$

If v is critical,

$$\begin{aligned} y(v) &\geq 1 + R - w(\delta^+_F(v)) + (1-R) + (\beta - 1) \\ &\geq \beta + z(\delta^+_F(v)) - w(\delta^+_F(v)) \geq z(C). \end{aligned}$$

CASE 4.2: $|\delta^+_Q(v)| = 1$. $\delta^+_Q(v)$ must be a regular flip by Fact 74. Also, $b_v \geq -(1-R)\cdot|\delta^-_F(v)| + (1 - 2R + w(\delta^+_Q(v)))$. The two cases $|\delta^-_H(v)| = 0$ and $|\delta^-_H(v)| = 1$ work nearly the same way, but for clarity first assume that $|\delta^-_H(v)| = 0$. This means that all edges $e \in \delta^-_{\tilde{E}}(v)$ have $z(e) \geq w(e)$. If v is not critical, we argue that

$$\begin{aligned} y(v) &\geq z(\delta^-(v)) - w(\delta^-_{\tilde{E}}(v)) \\ &\quad + 1 + R - w(\delta^+_Q(v)) - (1-R) + (1 - 2R + w(\delta^-_Q(v))) \\ &= z(\delta^-(v)) - w(\delta^-_{\tilde{E}}(v)) + 1 \geq z(C). \end{aligned}$$

Here we use that when $z(e) > w(e)$ then either e is tiny and therefore must be in $\delta^-(v)$ or else the flip e could be added to P or e is in $F \setminus Q$ and therefore also in $\delta^-(v)$. On the other hand, if v is critical, then

$$\begin{aligned} y(v) &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 1 + R - w(\delta^+_Q(v)) \\ &\quad - (1-R) + (1 - 2R + w(\delta^-_Q(v))) + (\beta - 1) \\ &= z(\delta^-_F(v)) - w(\delta^-_F(v)) + \beta \geq z(C). \end{aligned}$$

Now assume that $|\delta^-_H(v)| = 1$. In particular, $|\delta^-_F(v)| = 0$. If v is not critical, then

$$\begin{aligned} y(v) &\geq z(\delta^-_T(v)) - w(\delta^-_{\tilde{E}\cap T}(v)) \\ &\quad + w(\delta^-_{\tilde{E}}(v)) - (1-R)\cdot|\delta^-_H(v)| + (1 - 2R + w(\delta^-_Q(v))) \\ &\geq z(\delta^-_T(v)) - w(\delta^-_{\tilde{E}\cap T}(v)) \\ &\quad + 1 + R - w(\delta^+_Q(v)) + (w(\delta^-_Q(v)) - R) \\ &= z(\delta^-_T(v)) - w(\delta^-_{\tilde{E}\cap T}(v)) + 1 \geq z(C). \end{aligned}$$

Finally, if v is critical, then

$$\begin{aligned} y(v) &\geq w(\delta^-_{\tilde{E}}(v)) - (1-R)\cdot|\delta^-_H(v)| + (1-2R+w(\delta^-_Q(v))) + (\beta-1) \\ &\geq 1 + R - w(\delta^+_Q(v)) + (w(\delta^-_Q(v)) - R) + (\beta-1) = \beta \geq z(C). \end{aligned}$$

7.3.4 *Proof of Claim 71*

Recall, this claim is the inequality $z(C) \leq y(v)$ for the case of $|\delta^-_B(v)| = 2$.

Case 1: v is a bad vertex

Note that no pending flip can go towards a bad vertex, hence by definition of $F \subseteq P$ it holds that $\delta^+_F(v) = \emptyset$ and therefore $z(C) \leq z(\delta^-_F(v)) - w(\delta^-_F(v)) + \beta$. Moreover, all edges $e \in \delta^-_{E\setminus H}(v)$ have $z(e) \geq w(e)$. Thus,

$$\begin{aligned} y(v) &\geq z(\delta^-(v)) + b_v - \mu \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + w(\delta^-_{\tilde{E}}(v)) \\ &\quad \underbrace{-|\delta^-_H(v)|\cdot(1-R)}_{\leq z(\delta^-_H(v)) - w(\delta^-_H(v))} \underbrace{-|\delta^-_{B\setminus H}(v)|\cdot(1-R)}_{\leq b_v} - \mu \\ &> z(\delta^-_F(v)) - w(\delta^-_F(v)) + \underbrace{1 + R - 2\cdot(1-R) - \mu}_{\geq \beta} \geq z(C). \end{aligned}$$

Case 2: v is good and prospect vertex of a tiny flip

Note that by definition of F, $\delta^+_F(v)$ must be empty. Let e_T be the tiny edge with a flip towards v. For all $e \in \delta^-_{F\setminus Q}(v)$ we have $w(e) > 0.6$ or $w(e) > 1 - w(e_T) \geq R > 0.6$ (otherwise e would be in Q by Fact 73). We will make frequent use of the inequality $w(\delta^-_{\tilde{E}}(v)) > 1 + R - w(e_T) \geq R + 2/3$.

CASE 2.1: $|\delta^-_{(F\setminus Q)\cup H}(v)| = 0$. Note that

$$b_v \geq -\sum_{e\in\delta^-_{F\cap Q}(v)} [1 - 2R + w(e)] \geq 2\cdot(2R - 1.6).$$

Thus,

$$y(v) \geq w(\delta^-_{\tilde{E}}(v)) - (\beta-1) + 4R - 3.2 > R + 2/3 - \beta + 4R - 2.2 \geq \beta \geq z(C).$$

CASE 2.2: $|\delta^-_{(F\setminus Q)\cup H}(v)| = 1$. Then

$$\begin{aligned} y(v) &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + w(\delta^-_{\tilde{E}}(v)) \\ &\quad \underbrace{-|\delta^-_H(v)|\cdot(1-R)}_{\leq z(\delta^-_H(v)) - w(\delta^-_H(v))} \underbrace{-|\delta^-_{F\setminus Q}(v)|\cdot(1-R) - (1.6-2R)}_{\leq b_v} - (\beta-1) \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + R + \frac{2}{3} - (1-R) - 0.6 + 2R - \beta \\ &= z(\delta^-_F(v)) - w(\delta^-_F(v)) + 4R + \frac{2}{3} - 1.6 - \beta \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + \beta \geq z(C). \end{aligned}$$

CASE 2.3: $|\delta^-_{(F\setminus Q)\cup H}(v)| = 2$. Here we use that $z(C) \leq \beta$, if $C \cap (F \setminus Q) = \emptyset$ and $z(C) \leq 1 + \beta(1 - w(C \cap (F \setminus Q))) \leq 1 + 0.4\beta$, otherwise. This gives

$$\begin{aligned} y(v) \geq |\delta^-_{F\setminus Q}(v)|\cdot 1 + |\delta^-_H(v)|\cdot R - |\delta^-_{F\setminus Q}(v)|\cdot(1-R) - (\beta-1) &= 2R+1-\beta \\ \geq \max\{\beta, 0.4\beta+1\} &\geq z(C). \end{aligned}$$

Case 3: v is good, prospect vertex of a small flip, but not of a tiny flip

Note that by definition of F, $\delta^+_F(v)$ must be empty. Let e_S be the smallest edge with a flip towards v.

First, let us handle the case, where e_S is only a raw pending flip, i.e., $(e_S, \text{RAW}) \in P$, but $(e_S, \text{REG}) \notin P$. This is exactly the case when for both $e_B \in \delta^-_B(v)$ it holds that $w(e_B) + w(e_S) > 1$ (Fact 75). This implies that v only repels big edges and edges of weight greater than $w(e_S)$. Moreover, $\delta^-_Q(v) = \emptyset$ and therefore

$$\begin{aligned} y(v) \geq \sum_{e\in\delta^-_F(v)} [1-(1-R)] + \sum_{e\in\delta^-_{B\setminus F}(v)} w(e) + z(\delta^-_{\tilde{E}\setminus B}(v)) \\ \geq 1 + z(\delta^-_{\tilde{E}\setminus B}(v)) \geq z(C). \end{aligned}$$

We can therefore assume what $(e_S, \text{REG}) \in P$.

CASE 3.1: $|\delta^-_{(F\setminus Q)\cup H}(v)| = 0$. We bound b_v by

$$b_v \geq -\sum_{e\in F\cap Q} [1 + 2R - w(e)] \geq 4R - 3.2.$$

Thus,

$$y(v) \geq w(\delta^-_{\tilde{E}}(v)) + 4R - 3.2 \geq 1 + R - 0.5 + 4R - 3.2 \geq \beta \geq z(C).$$

CASE 3.2: $|\delta^-_{(F\setminus Q)\cup H}(v)| = 1$ AND v IS UNCRITICAL. Note that if $\delta^-_{F\cap Q}(v) = \emptyset$, then

$$z(\delta^-_B(v)) + b_v \geq R + 0.5 \geq 3R - 1.$$

On the other hand, when $|\delta^-_{F\cap Q}(v)| = 1$, then

$$z(\delta^-_B(v)) + b_v \geq R + w(\delta^-_{F\cap Q}(v)) - (1 - 2R + w(\delta^-_{F\cap Q}(v))) = 3R - 1.$$

If C does not contain an edge from $F \setminus Q$,

$$y(v) \geq 3R - 1 \geq \beta \geq z(C).$$

Assume in the remainder of Case 3.2 that C contains an edge from $\delta^-_{F\setminus Q}(v) \neq \emptyset$. If C contains no small edge from $\tilde{E} \setminus \delta^-(v)$

$$y(v) \geq z(\delta^-_B(v)) + z(\delta^-_{\tilde{E}\setminus B}(v)) + b_v \geq 3R - 1 + z(\delta^-_{\tilde{E}\setminus B}(v)) \geq z(C).$$

Assume now that C contain $\delta^-_{F\setminus Q}(v)$ and a small edge $e \in \tilde{E} \setminus \delta^-(v)$. This implies $w(\delta^-_{F\setminus Q}(v)) > 0.6$, since $w(\delta^-_{F\setminus Q}(v)) + w(e) \leq w(C) \leq 1$ and the edge would be in Q otherwise. If $\delta^-_{\tilde{E}}(v)$ contains a small edge,

$$y(v) \geq z(\delta^-_B(v)) + z(\delta^-_{\tilde{E}\setminus B}(v)) + b_v \geq 3R - 1 + 1/3 \geq 1 + \beta(1 - 0.6) \geq z(C).$$

Assume now $\delta^-_{\tilde{E}}(v)$ contains no small edge. Then $w(e) \geq w(e_S)$ or else the edge would not be repelled by v. This implies $w(e_S) \leq w(e) \leq 1 - w(\delta^-_{F\setminus Q}(v)) < 0.4$. If $|\delta^-_{F\cap Q}(v)| = 1$,

$$\begin{aligned}
y(v) &\geq z(\delta^-(v)) + b_v \\
&\geq z(\delta^-_B(v)) + \beta(w(\delta^-_{\tilde{E}}(v)) - w(\delta^-_B(v))) + b_v \\
&\geq 3R - 1 + \beta(1 + R - w(e_S) - w(\delta^-_B(v))) \\
&= 3R - 1 + \beta(1 - w(e_S) - w(\delta^-_{F\setminus Q}(v))) + \beta(R - w(\delta^-_{F\cap Q}(v))) \\
&\geq \underbrace{3R - 1 + \beta(R - 0.6)}_{\geq 1.4 \geq z(\delta^-_{F\setminus Q}(v)) + z(e)} + \beta(1 - w(e) - w(\delta^-_{F\setminus Q}(v))) \geq z(C).
\end{aligned}$$

If $|\delta^-_{F\cap Q}(v)| = 0$,

$$\begin{aligned}
y(v) &\geq z(\delta^-(v)) + b_v \geq z(\delta^-_B(v)) + \beta(w(\delta^-_{\tilde{E}}(v)) - w(\delta^-_B(v))) + b_v \\
&\geq R + w(\delta^-_{B\setminus F}(v)) + \beta(1 + R - w(e_S) - w(\delta^-_B(v))) \\
&= R + w(\delta^-_{B\setminus F}(v)) + \beta(R - w(\delta^-_{B\setminus F}(v))) \\
&\quad + \beta(1 - w(e_S) - w(\delta^-_F(v))) \\
&\geq \underbrace{R + R + \beta(R - R)}_{\geq 1.4} + \beta(1 - w(e) - w(\delta^-_F(v))) \geq z(C).
\end{aligned}$$

Here we use that $w(\delta^-_{B\setminus F}(v)) \leq R$, since it is not in H.

CASE 3.3: $|\delta^-_{(F\setminus Q)\cup H}(v)| = 1$, $|\delta^-_{F\cap Q}(v)| = 0$ AND v IS CRITICAL. If C contains $\delta^-_{F\setminus Q}(v) \neq \emptyset$,

$$\begin{aligned}
y(v) &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 1 + R - w(e_S) - (1 - R) + (\beta - 1) \\
&\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 2R - 1.5 + \beta \\
&\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 0.5 + 0.5\beta \geq z(C).
\end{aligned}$$

If $C \cap F = \emptyset$,

$$y(v) \geq R + w(\delta^-_{B\setminus(F\cup H)}(v)) + (\beta - 1) \geq R + \beta - 0.5 \geq \beta \geq z(C),$$

since there must be a second big edge on v.

CASE 3.4: $|\delta^-_{(F\setminus Q)\cup H}(v)| = 1$, $|\delta^-_{F\cap Q}(v)| = 1$ AND v IS CRITICAL. Since e_S is a regular pending flip, there must be an edge $e_B \in \delta^-_B(v)$ with $w(e_S) + w(e_B) \leq 1$. This implies $w(e_S) + w(\delta^-_{F\cap Q}(v)) \leq 1$: If $e_B \in \delta^-_{F\setminus Q}(v)$, it must be that $w(e_B) > 0.6$. Therefore, $w(e_S) + w(\delta^-_{F\cap Q}(v)) \leq w(e_S) + 0.6 < w(e_S) + w(e_B) \leq 1$. If C contains $\delta^-_{F\setminus Q}(v) \neq \emptyset$,

$$\begin{aligned} y(v) &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 1 + R - w(e_S) - (1 - R) \\ &\quad - (1 - 2R + w(\delta^-_{F\cap Q}(v))) + (\beta - 1) \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 4R - 2 + \beta \\ &\quad - (w(\delta^-_{F\cap Q}(v)) + w(e_S)) \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 4R - 3 + \beta \\ &\geq z(\delta^-_F(v)) - w(\delta^-_F(v)) + 0.5 + 0.5\beta \geq z(C). \end{aligned}$$

If $C \cap (F \setminus Q) = \emptyset$,

$$\begin{aligned} y(v) &\geq R + w(\delta^-_{F\cap Q}(v)) - (1 - 2R + w(\delta^-_{F\cap Q}(v))) + (\beta - 1) \\ &\geq 3R - 2 + \beta \geq \beta \geq z(C). \end{aligned}$$

CASE 3.5: $|\delta^-_{(F\setminus Q)\cup H}(v)| = 2$ AND $|\delta^-_{F\cap Q}(v)| = 0$. Let $e_B \in \delta^-_B(v)$ be the smaller of the two big edges. Since e_S is a regular pending flip, $w(e_S) + w(e_B) \leq 1$. Therefore $w(e_B) \leq 2/3$ and e_B cannot be in H. Since it is then in $F \setminus Q$, it must be greater than 0.6. In particular, all edges in $\delta^-_B(v)$ are greater than 0.6. If C does not contain an edge from $F \setminus Q$, then $y(v) \geq 2R \geq \beta \geq z(C)$. Otherwise, $y(v) \geq 2R \geq 1 + 0.4\beta \geq z(C)$.

Case 4: v is good and is not prospect vertex of a small/tiny flip

Note that $|\delta^+_F(v)| \leq 1$. If $|\delta^+_F(v)| = 0$, then v does not repel any edges. Thus, $b_v = 0$ and for every $e \in \delta(v)$ with $z(e) > 0$ it holds that $e \in \delta^-(v)$. Therefore, $z(C) \leq z(\delta^-(v)) = y(v)$. Assume now $|\delta^+_F(v)| = 1$. v repels only big edges, since it cannot be prospect vertex of a regular flip. Thus, $\delta^-_Q(v) = \emptyset$ and

$$\begin{aligned} y(v) &\geq R \cdot |\delta^-_{F\cup H}(v)| + 0.5 \cdot |\delta^-_{B\setminus(F\cup H)}(v)| + z(\delta^-_{\tilde{E}\setminus B}(v)) \\ &\geq 1 + z(\delta^-_{\tilde{E}\setminus B}(v)) \geq z(C). \end{aligned}$$

BIBLIOGRAPHY

[1] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. "SETH-Based Lower Bounds for Subset Sum and Bicriteria Path." In: *CoRR* abs/1704.04546 (2017).

[2] Iskander Aliev, J.A. De Loera, Fritz Eisenbrand, Timm Oertel, and Robert Weismantel. "The support of integer optimal solutions." In: *SIAM Journal on Optimization* 28.3 (2018), pp. 2152–2157.

[3] Noga Alon, Raphael Yuster, and Uri Zwick. "Color-coding." In: *Journal of the ACM* 42.4 (1995). Previously appeared in STOC 1994, pp. 844–856.

[4] Katerina Altmanová, Dušan Knop, and Martin Koutecký. "Evaluating and Tuning n-fold Integer Programming." In: *17th International Symposium on Experimental Algorithms*. Vol. 103. Leibniz International Proceedings in Informatics (LIPIcs). Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 10:1–10:14.

[5] Alexandr Andoni and Krzysztof Onak. "Approximating edit distance in near-linear time." In: *SIAM Journal on Computing* 41.6 (2012). Previously appeared in STOC 2009, pp. 1635–1648.

[6] Chidambaram Annamalai. "Lazy Local Search Meets Machine Scheduling." In: *CoRR* abs/1611.07371 (2016).

[7] Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. "Combinatorial Algorithm for Restricted Max-Min Fair Allocation." In: *ACM Trans. Algorithms* 13.3 (2017), 37:1–37:28.

[8] Arash Asadpour, Uriel Feige, and Amin Saberi. "Santa claus meets hypergraph matchings." In: *ACM Trans. Algorithms* 8.3 (2012). See Asadpour's homepage for the bound of 4 instead of 5 as in the paper, 24:1–24:9.

[9] Yuichi Asahiro, Jesper Jansson, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. "Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree." In: *Journal of Combinatorial Optimization* 22.1 (2011), pp. 78–96.

[10] Kyriakos Axiotis and Christos Tzamos. "Capacitated Dynamic Programming: Faster Knapsack and Graph Algorithms." In: *46th International Colloquium on Automata, Languages, and Programming, ICALP'19, Patras, Greece, July 8-12, 2019*. 2019.

[11] Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. "Better Approximations for Tree Sparsity in Nearly-Linear Time." In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. 2017, pp. 2215–2229.

[12] Nikhil Bansal and Maxim Sviridenko. “The Santa Claus problem.” In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*. 2006, pp. 31–40.

[13] József Beck and Tibor Fiala. “"Integer-making" theorems.” In: *Discrete Applied Mathematics* 3.1 (1981), pp. 1–8.

[16] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. “Necklaces, Convolutions, and X+Y.” In: *Algorithmica* 69.2 (2014), pp. 294–314.

[17] Karl Bringmann. “A Near-Linear Pseudopolynomial Time Algorithm for Subset Sum.” In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. 2017, pp. 1073–1084.

[18] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. “On Allocating Goods to Maximize Fairness.” In: *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*. 2009, pp. 107–116.

[19] Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. “On $(1,\epsilon)$-Restricted Assignment Makespan Minimization.” In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*. 2015, pp. 1087–1101.

[20] Deeparnab Chakrabarty and Kirankumar Shiragur. “Graph Balancing with Two Edge Types.” In: *CoRR* abs/1604.06918 (2016).

[21] Timothy M. Chan and Moshe Lewenstein. “Clustered Integer 3SUM via Additive Combinatorics.” In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. 2015, pp. 31–40.

[22] Siu-Wing Cheng and Yuchen Mao. “Integrality Gap of the Configuration LP for the Restricted Max-Min Fair Allocation.” In: *CoRR* abs/1807.04152 (2018).

[23] Siu-Wing Cheng and Yuchen Mao. “Restricted Max-Min Allocation: Approximation and Integrality Gap.” In: *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*. 2019, 38:1–38:13.

[24] William Cook, Jean Fonlupt, and Alexander Schrijver. “An integer analogue of Carathéodory’s theorem.” In: *Journal of Combinatorial Theory, Series B* 40.1 (1986), pp. 63–70.

[25] Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. “On Problems Equivalent to (min, +)-Convolution.” In: *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*. 2017, 22:1–22:15.

[26] Jesús A. De Loera, Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. "N-fold integer programming." In: *Discrete Optimization* 5.2 (2008), pp. 231–241.

[27] Doratha E. Drake and Stefan Hougardy. "A linear time approximation algorithm for weighted matchings in graphs." In: *Journal of the ACM Transactions on Algorithms* 1.1 (2005), pp. 107–122.

[28] Tomás Ebenlendr, Marek Krcál, and Jiri Sgall. "Graph Balancing: A Special Case of Scheduling Unrelated Parallel Machines." In: *Algorithmica* 68.1 (2014), pp. 62–80.

[29] Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. "Faster Algorithms for Integer Programs with Block Structure." In: *45th International Colloquium on Automata, Languages, and Programming*. Vol. 107. Leibniz International Proceedings in Informatics (LIPIcs). Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 49:1–49:13.

[30] Friedrich Eisenbrand and Robert Weismantel. "Proximity results and faster algorithms for Integer Programming using the Steinitz Lemma." In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. 2018, pp. 808–816.

[31] Fedor V. Fomin, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. "On the Optimality of Pseudo-polynomial Algorithms for Integer Programming." In: *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*. 2018, 31:1–31:13.

[32] Michael L. Fredman, János Komlós, and Endre Szemerédi. "Storing a Sparse Table with $O(1)$ Worst Case Access Time." In: *Journal of the ACM* 31.3 (1984). Previously appeared in FOCS 1982, pp. 538–544.

[33] Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. "N-fold integer programming in cubic time." In: *Mathematical Programming* (2013), pp. 1–17.

[34] Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. "N-fold integer programming and nonlinear multi-transshipment." In: *Optimization Letters* 5.1 (2011), pp. 13–25.

[35] Chien-Chung Huang and Sebastian Ott. "A Combinatorial Approximation Algorithm for Graph Balancing with Light Hyper Edges." In: *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*. 2016, 49:1–49:15.

[36] Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. "Empowering the Configuration-IP - New PTAS Results for Scheduling with Setups Times." In: *ITCS*. Vol. 124. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, 44:1–44:19.

[37] Klaus Jansen, Kim-Manuel Klein, and José Verschae. "Closing the Gap for Makespan Scheduling via Sparsification Techniques." In: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*. 2016, 72:1–72:13.

[38] Klaus Jansen, Kati Land, and Marten Maack. "Estimating The Makespan of The Two-Valued Restricted Assignment Problem." In: *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*. 2016, 24:1–24:13.

[39] Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. "Near-Linear Time Algorithm for n-fold ILPs via Color Coding." In: *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*. 2019, 75:1–75:13.

[40] Klaus Jansen and Lars Rohwedder. "A Quasi-Polynomial Approximation for the Restricted Assignment Problem." In: *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization, IPCO'17, Waterloo, ON, Canada, June 26-28, 2017*. 2017, pp. 305–316.

[41] Klaus Jansen and Lars Rohwedder. "On the Configuration-LP of the Restricted Assignment Problem." In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'17, Barcelona, Spain, January 16-19*. 2017, pp. 2670–2678.

[43] Klaus Jansen and Lars Rohwedder. "A note on the integrality gap of the configuration LP for restricted Santa Claus." In: *CoRR* abs/1807.03626 (2018).

[44] Klaus Jansen and Lars Rohwedder. "Compact LP Relaxations for Allocation Problems." In: *1st Symposium on Simplicity in Algorithms, SOSA'18, January 7-10, 2018, New Orleans, LA, USA*. 2018, 11:1–11:19.

[45] Klaus Jansen and Lars Rohwedder. "Local Search Breaks 1.75 for Graph Balancing." In: *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*. 2019, 74:1–74:14.

[46] Klaus Jansen and Lars Rohwedder. "On Integer Programming and Convolution." In: *10th Innovations in Theoretical Computer Science Conference, ITCS'19, January 10-12, 2019, San Diego, California, USA*. 2019, 43:1–43:17.

[47] Hendrik W. Lenstra Jr. "Integer Programming with a Fixed Number of Variables." In: *Math. Oper. Res.* 8.4 (1983), pp. 538–548.

[48] Ravi Kannan. "Minkowski's Convex Body Theorem and Integer Programming." In: *Math. Oper. Res.* 12.3 (1987), pp. 415–440.

[49] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

[50] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. "An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations." In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 217–226.

[51] Dušan Knop and Martin Koutecký. "Scheduling meets n-fold integer programming." In: *Journal of Scheduling* (2017), pp. 1–11.

[52] Dušan Knop, Martin Koutecký, and Matthias Mnich. "Combinatorial n-fold Integer Programming and Applications." In: *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*. 2017, 54:1–54:14.

[53] Dusan Knop, Michal Pilipczuk, and Marcin Wrochna. "Tight Complexity Lower Bounds for Integer Linear Programming with Few Constraints." In: *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*. 2019, 44:1–44:15.

[54] Martin Koutecký, Asaf Levin, and Shmuel Onn. "A Parameterized Strongly Polynomial Algorithm for Block Structured Integer Programs." In: *45th International Colloquium on Automata, Languages, and Programming*. Vol. 107. Leibniz International Proceedings in Informatics (LIPIcs). Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 85:1–85:14.

[55] Eduardo Sany Laber, Wilfredo Bardales Roncalla, and Ferdinando Cicalese. "On lower bounds for the Maximum Consecutive Subsums Problem and the (min, +)-convolution." In: *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014*. 2014, pp. 1807–1811.

[56] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. "Approximation Algorithms for Scheduling Unrelated Parallel Machines." In: *Math. Program.* 46 (1990), pp. 259–271.

[57] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. "Splitters and Near-Optimal Derandomization." In: *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*. 1995, pp. 182–191.

[58] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. "Splitters and near-optimal derandomization." In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE. 1995, pp. 182–191.

[59] Shmuel Onn and Pauline Sarrabezolles. "Huge Unimodular n-Fold Programs." In: *SIAM J. Discrete Math.* 29.4 (2015), pp. 2277–2283.

[60] Daniel R. Page and Roberto Solis-Oba. "A 3/2-Approximation Algorithm for the Graph Balancing Problem with Two Weights." In: *Algorithms* 9.2 (2016), p. 38.

[61] Christos H. Papadimitriou. "On the complexity of integer programming." In: *J. ACM* 28.4 (1981), pp. 765–768.

[62] Lukás Polácek and Ola Svensson. “Quasi-Polynomial Local Search for Restricted Max-Min Fair Allocation.” In: *ACM Trans. Algorithms* 12.2 (2016), 13:1–13:13.

[63] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999. ISBN: 978-0-471-98232-6.

[64] Petra Schuurman and Gerhard J. Woeginger. “Polynomial time approximation algorithms for machine scheduling: ten open problems.” In: *Journal of Scheduling* 2.5 (1999).

[65] Sergey V. Sevastyanov. “Approximate solution of some problems in scheduling theory.” In: *Metody Diskret. Analiz* 32 (1978). in Russian, pp. 66–75.

[66] Joel Spencer. “Six standard deviations suffice.” In: *Transactions of the American mathematical society* 289.2 (1985), pp. 679–706.

[67] Ernst Steinitz. “Bedingt konvergente Reihen und konvexe Systeme.” In: *Journal für die reine und angewandte Mathematik* 143 (1913), pp. 128–176.

[68] Ola Svensson. “Santa Claus Schedules Jobs on Unrelated Machines.” In: *SIAM J. Comput.* 41.5 (2012), pp. 1318–1341.

[69] Éva Tardos. “A Strongly Polynomial Algorithm to Solve Combinatorial Linear Programs.” In: *Operations Research* 34.2 (1986), pp. 250–256.

[70] José Verschae and Andreas Wiese. “On the configuration-LP for scheduling on unrelated machines.” In: *J. Scheduling* 17.4 (2014), pp. 371–383.

[71] Chao Wang and René Sitters. “On some special cases of the restricted assignment problem.” In: *Inf. Process. Lett.* 116.11 (2016), pp. 723–728.

LARS ROHWEDDER - CURRICULUM VITAE

DATE OF BIRTH July 1st, 1991

PLACE OF BIRTH Rendsburg, Germany

CITIZENSHIP German

EDUCATION

- Ph.D. Candidate, Advisor Klaus Jansen, Feb. 2016- , University of Kiel, Germany
- Master of Science in Computer Science, Mar. 2013 - Feb. 2016 (6 semesters), University of Kiel, Germany
- Bachelor of Science in Computer Science, Okt. 2010 - Feb. 2013 (5 semesters), University of Kiel, Germany
- Abitur at Helene-Lange-Gymnasium, Rendsburg, Germany

INTERNSHIPS

- Research Assistant at VMWare, Apr. 2015 - June 2015, Palo Alto, USA
- Research Assistant at Oracle Labs, Feb. 2014 - Apr. 2014, Redwood Shores, USA
- Research Assistant at Oracle Labs, Mar. 2013 - Sep. 2013, Redwood Shores, USA

REVIEWING ACTIVITIES (SELECTION)

- STOC, SODA, ICALP, ESA, WAOA, COCOON
- SIAM Journal on Computing, Theoretical Computer Science

CONFERENCE PRESENTATIONS

- For the papers [44, 40, 41, 46, 42]

PUBLICATIONS AND MANUSCRIPTS

[14] Sebastian Berndt, Leah Epstein, Klaus Jansen, Asaf Levin, Marten Maack, and Lars Rohwedder. "Online Bin Covering with Limited Migration." In: *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.* 2019, 18:1–18:14.

[15] Marin Bougeret, Klaus Jansen, Michael Poss, and Lars Rohwedder. "Approximation results for makespan minimization with budgeted uncertainty." In: *Approximation and Online Algorithms - 17th International Workshop, WAOA 2018, Munich/Garching, Germany, September 12-13, 2019, Revised Selected Papers*. to appear. 2019.

[39] Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. "Near-Linear Time Algorithm for n-fold ILPs via Color Coding." In: *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*. 2019, 75:1–75:13.

[40] Klaus Jansen and Lars Rohwedder. "A Quasi-Polynomial Approximation for the Restricted Assignment Problem." In: *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization, IPCO'17, Waterloo, ON, Canada, June 26-28, 2017*. 2017, pp. 305–316.

[41] Klaus Jansen and Lars Rohwedder. "On the Configuration-LP of the Restricted Assignment Problem." In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'17, Barcelona, Spain, January 16-19*. 2017, pp. 2670–2678.

[42] Klaus Jansen and Lars Rohwedder. "Structured Instances of Restricted Assignment with Two Processing Times." In: *Algorithms and Discrete Applied Mathematics - Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16-18, 2017, Proceedings*. 2017, pp. 230–241.

[43] Klaus Jansen and Lars Rohwedder. "A note on the integrality gap of the configuration LP for restricted Santa Claus." In: *CoRR* abs/1807.03626 (2018).

[44] Klaus Jansen and Lars Rohwedder. "Compact LP Relaxations for Allocation Problems." In: *1st Symposium on Simplicity in Algorithms, SOSA'18, January 7-10, 2018, New Orleans, LA, USA*. 2018, 11:1–11:19.

[45] Klaus Jansen and Lars Rohwedder. "Local Search Breaks 1.75 for Graph Balancing." In: *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*. 2019, 74:1–74:14.

[46] Klaus Jansen and Lars Rohwedder. "On Integer Programming and Convolution." In: *10th Innovations in Theoretical Computer Science Conference, ITCS'19, January 10-12, 2019, San Diego, California, USA*. 2019, 43:1–43:17.

ERKLÄRUNG

Diese Abhandlung ist nach Inhalt und Form meine eigene Arbeit. Ausnahmen sind die Beratung durch meinen Betreuer Prof. Dr. Klaus Jansen sowie teilweise Kapitel 4, welches in Zusammenarbeit mit Alexandra Lassota entstanden ist, und Kapitel 3, dessen Ergebnisse zusammen mit José Verschae erarbeitet worden sind.

Diese Dissertation wurde weder ganz noch in Teilen an anderer Stelle im Rahmen eines Prüfungsverfahrens vorgelegt. Die Veröffentlichungen und Einreichungen zur Veröffentlichung, auf denen die Dissertation basiert, sind in Seite vii aufgeführt.

Die Arbeit ist unter Einhaltung der Regeln guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft entstanden.

Kiel, September 2019

Lars Rohwedder

COLOPHON

The drawings at the beginning and end of the chapters were created by freelance illustrator Nora Grunwald and are subject to

https://www.noragrunwald.com/

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić with some modifications by the author. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Final Version as of September 26, 2019 (classicthesis v4.6).